Analysis of Security Protocols in Embedded Systems

Bruni, Alessandro

Publication date:
2016

Document Version
Publisher's PDF, also known as Version of record

Analysis of Security Protocols in Embedded Systems

Alessandro Bruni

Kongens Lyngby 2015

Technical University of Denmark
Department of Applied Mathematics and Computer Science
Richard Petersens Plads, building 324,
2800 Kongens Lyngby, Denmark
Phone +45 4525 3031
compute@compute.dtu.dk
www.compute.dtu.dk

Summary (English)

Embedded real-time systems have been adopted in a wide range of safety-critical applications — including automotive, avionics, and train control systems — where the focus has long been on *safety* (i.e., protecting the external world from the potential damage caused by the system) rather than *security* (i.e., protecting the system from the external world). With increased connectivity of these systems to external networks the attack surface has grown, and consequently there is a need for securing the system from external attacks. Introducing security protocols in safety critical systems requires careful considerations on the available resources, especially in meeting real-time and resource constraints, as well as cost and reliability requirements. For this reason many proposed security protocols in this domain have peculiar features, not present in traditional security literature.

In this thesis we tackle the problem of analysing security protocols in safety critical embedded systems from multiple perspectives, extending current state-of-the-art analysis techniques where the combination of safety and security hinders our efforts. Examples of protocols in automotive control systems will follow throughout the thesis. We initially take a combined perspective of the safety and security features, by giving a security analysis and a schedulability analysis of the embedded protocols, with intertwined considerations. Then we approach the problem of the expressiveness of the tools used in the analysis, extending saturation-based techniques for formal protocol verification in the symbolic model. Such techniques gain much of their efficiency by coalescing all reachable states into a single set of facts. However, distinguishing different states is a requirement for modelling the protocols that we consider. Our effort in this direction is to extend saturation-based techniques so that enough state information can be modelled and analysed. Finally, we present a methodology for proving the same security properties in the computational model, by means of typing protocol implementations.

Summary (Danish)

Indlejrede realtidssystemer har fundet anvendelse på en lang række sikkerhedskritiske områder, såsom i bilindustrien, flyindustrien og kontrolsystemer til tog. På disse områder har fokus længe været på *pålidelighed* (dvs. at beskytte omverden mod potentiel skade forårsaget af systemet), snarere end *sikkerhed* (dvs. at beskytte systemet mod omverden). Med tættere forbindelser mellem disse systemer og eksterne netværk, er angrebsfladen blevet større. Som konsekvens, er der brug for beskyttelse mod angreb udefra. At benytte sikkerhedsprotokoller i kritiske systemer, kræver nøje afvejninger af de tilgængelige ressourcer, især når krav til realtid og ressourceforbrug skal overholdes, samtidig med at prisen ikke må stige og pålideligheden forbliver høj. Af denne grund indeholder mange foreslåede sikkerhedsprotokoller utraditionelle løsninger, der ikke er at finde i den klassiske sikkerhedslitteratur.

I denne afhandling tackler vi problemet med at analysere protokoller til sikkerhedskritiske indlejrede systemer fra flere forskellige vinkler. Vi udvider de eksisterende analyseteknikker hvor kombinationen af sikkerhed og pålidelighed gør dem utilstrækkelige. Gennem afhandlingen vil protokoller fra bilindustrien løbende blive brugt som eksempler. Vi betragter som udgangspunkt sikkerhed og pålidelighed fra et samlet perspektiv, ved at give en sikkerheds- og skeduleringesanalyse af de indlejrede protokoller. Dernæst nærmer vi os problemet med udtryksfuldheden for de værktøjer der blev brugt til analysen. Vi udvider de mætningsbaserede teknikker til formel protokolverifikation i den symbolske model. Disse teknikker er meget effektive på grund af evnen til at kombinere alle tilgængelige tilstande til ét sæt fakta. At kunne skelne mellem tilstande, er dog et krav for at kunne modellelere de protokoller vi gerne vil analysere. Dette problem løser vi ved at udvide de mætningsbaserede teknikker, så der gemmes tilstrækkelig tilstand til at protokollerne kan modelleres og analyseres. Endelig præsenterer vi en metode til at bevise de samme sikkerhedsegenskaber i den beregningsmæssige model ved hjælp af type-annoterede protokolimplementationer.

Preface

This thesis was prepared at DTU Compute, the Department of Applied Mathematics and Computer Science of the Technical University of Denmark, in partial fulfilment of the requirements for acquiring the Ph.D. degree in Computer Science.

The Ph.D. study has been carried out under the supervision of Professor Flemming Nielson and Professor Hanne Riis Nielson in the period between October 2012 and September 2015. The Ph.D. project is funded by SESAMO, European ARTEMIS Project nr. 295354.

Most of the work behind this dissertation has been carried out independently and I take full responsibility for its contents. Part of the scientific work reported in this thesis is based on joint work with my supervisors, and in collaboration with Michal Sojka from the Czech Technical University [BSNN14], and Sebastian Mödersheim [BMNN15].

My research visit of Professor Mark Ryan and Eike Ritter at the University of Birmingham, and Myrto Arapinis at the University of Edinburgh, besides insightful discussions and studies on the expressive power of the StatVerif and Set-Pi calculi and their analyses, inspired collaboration with Markulf Kohlweiss, which resulted in a workshop submission [BKA+15], now expanded to Chapter 7 of this thesis.

Kongens Lyngby,
30 September 2015

Alessandro Bruni

Acknowledgements

First of all I should like to thank my supervisors, Professor Flemming Nielson and Professor Hanne Riis Nielson, for bearing with my temper and supporting me throughout these studies. I am truly grateful for the opportunities they have given me. Then I would like to thank Sebastian Mödersheim, for transmitting his passion and motivation, and for showing me that there is a use in what we are doing. Umglaublich! I am also very grateful to Michal Sojka, for being my partner in crime, as I would not know how to touch a microcontroller, and for his hospitality both in formal and informal occasions. Speaking of hospitality, special thanks go to Professor Mark Ryan, Eike Ritter and Myrto Arapinis, for their time and dedication during my external stay, for transmitting their passion and for their joyful extravagance, each in their own way. I would also like to thank Markulf Kohlweiss, for his time and dedication in reviewing my work.

I would not have undertaken this endeavour if it wasn't for the passion transmitted by Professor Paolo Baldan, who first introduced me to formal methods, Professor Gilberto Filè, always pushing me to new lands and experiences, and Roberto, whose path I've grown accustomed to follow, by chance and vocation. My gratitude goes to them.

Among those who have lightened my days in the office, I would like to thank: Roberto, for looking always at the bright side of life, and for being a great source of inspiration; Zara, for her reserved and motherly attentions; Marieta, for her enthusiasm on every little thing, and the lack thereof on the big one; Laust, because, one day, we will succeed in one of our side-projects; Lars, for being present when the coffee machine breaks, and for our exciting disagreements; Omar, for always sharing his positive thoughts; Alberto, for showing that it's possible to grow old, without growing up; Christian, for his true German style and humour; Michal, for teaching me how to be lazy even in programming; Jan, for showing how everything is a Galois connection, and for his

passion and dedication to research and family; Ender and Lijun, for their passion and dedication to family and research; Erisa, for her overwhelming excitement; Andrea, for his placid and good-natured company; Hugo, for his life is a beautiful coreography; Natalya and Ximeng, for being quiet but friendly companions. I should also be grateful to the present and former secretaries, Cathrin and Lotte, for relieving me from many administrative duties, and reminding me of the ones I should carry out myself. I would like to thank my colleagues from Birmingham: Loretta, for her support and guidance in my brief experience; Mike, Jiangshan, Matt, Rusen, Luca, Bram and Guru for keeping good company, for the great movie nights, brewing nights and hacking sessions; and Dave, for our many failed attempts to meet. Among those whom I have encountered in this path and mentioned in this page, there are some who have been my hiking companions. I am truly, genuinely grateful to them for reaching together the high peaks that are so hard to come by in this country.

Much of my love and gratitude goes to my family and close friends. They have given me the love and support that I most cherished through the merry and the tough moments of my life, and continue to do so unconditionally. Very special thanks to Elena. Grazie per avermi seguito in quest'avventura e grazie per quelle che verranno, grazie per sopportare i miei sbalzi d'umore, grazie per supportarmi e per vivere ogni giornata accanto a me. Insomma, grazie d'esistere.

Contents

List of Figures

CHAPTER 1

Introduction

Distributed computing systems permeate the fabric of our society. When ARPANET — the ancestor of the modern Internet — was originally conceived in 1969, it sprung off from J.C.R. Licklider's vision of a "Galactic Computer Network". Licklider envisioned a global network of interconnected computers where every piece of data and program would be quickly accessible from any of its nodes.

Even though the concept of a "Galactic Computer Network" could have been compared to the dimensions of ARPANET, it cannot hold any comparison with the scale of the Internet today, let alone the Internet we are building now. A study published by CISCO [Eva11] calculated that, in 2010, 12.5 billion devices — almost twice the amount of living humans — were connected to the Internet. This number is growing at an exponential rate, doubling every five years.

Today's Internet does not only support the sharing of knowledge: it also runs banking and e-commerce applications, aids business and government operations worldwide. In its continuous transformation, the global network is now connecting more and more of the physical world, following the grand vision of the Internet of Things. Besides the proportions, which Licklider and his fellows would have probably never guessed, the original open design of ARPANET does not fit the bill of the current Internet.

Nowadays we need to protect from malicious attackers the data and programs that run through this globalised network. This need is going to increase in the future, as we build

more applications that rely on the Internet, and more real-world assets are at stake.

Security protocols play a central role in protecting the information and applications that run through the network from unwanted attacks. Designing security protocols is known to be a delicate and error prone process, and we know by experience that mistakes can go unnoticed for decades [Low96].

Systematically reasoning about the correctness of security protocols is therefore important to design secure systems. Formal methods provide theoretical frameworks and analysis techniques that can be used to reason about security properties in communication protocols. Formal approaches to security have proven their value in discovering flaws in existing designs, and are being used to ensure that new designs are immune to classes of attacks.

This thesis deals with the study and extension of formal methods for protocol verification. As the Internet expands to connect more of our physical world, new protocol designs are adopted that differ from the established literature, because physical constraints and safety requirements come into play. We study how these new designs can be described with formal languages and their peculiar properties captured and precisely analysed by means of automated verification techniques.

1.1 Challenge

Formal verification of security protocols has flourished over the last thirty years. The Dolev-Yao model [DY83], first proposed in 1983, is one formal model for describing security protocols where the security of many cryptographic functions is represented in symbolic terms. It allows formal automated reasoning about security protocols, and is used in many theoretical frameworks with tool support [Bla02, BBD+05, BMV05, Cre08, SMCB13]. Although the Dolev-Yao model is very successful for its simplicity, the problem of verifying security protocols in the Dolev-Yao model is in general undecidable [RD82], and continues to challenge the research community who tries to push the boundaries of machine-checked automated proofs.

Another model for describing protocols and their security properties is the computational model [Sho04]. In this model security protocols are described as probabilistic programs (called games), and cryptographic functions are treated as functions on bitstrings. It allows to capture a wider range of details in security protocols, at the cost of being more complex to reason about. Formal frameworks with semi- and fully-automated reasoning support are emerging [BGHB11, Bla07, FKS11], and showing promising results.

Both the symbolic and the computational model can prove various confidentiality

and integrity properties among the standard CIA triad: Confidentiality, Integrity and Availability. Confidentiality properties encompass *secrecy* (i.e. an attacker cannot obtain a piece of information) and *indistinguishability* (i.e. an attacker cannot distinguish whether two secrets are indeed the same). Integrity properties regard the integrity of data [Low97]. *Agreement* or *correspondence* is an integrity property that requires honest principals to agree on a specific message that they exchange, hence the attacker cannot alter the information without them noticing. A stronger notion of *agreement* is *injective-agreement* or *correspondence*, where an attacker also cannot reuse a message once that is consumed.

Availability is also an interesting category, as it regards the ability of a system to resist an attacker who aims to take it down. However, neither the symbolic nor the computational model can be used to reason about availability: they assume that the attacker is in control of all communication, hence the system can trivially be unavailable. A complementary approach is to study the robustness of a system under attack, and language approaches with optional data types can support this kind of analysis [VNN15].

As mentioned before, we have seen a trend towards interconnecting physical devices with each other, and more recently connecting them to the Internet. However, the symbolic and computational models have been mostly used for studying Internet protocols, hence there is an urgency to investigate their applicability to embedded systems. As an example that will follow us throughout this thesis, personal vehicles are integrating features like remote control and anti-theft protection, vehicle-to-vehicle communication, autonomous driving, etc., which all require network connectivity to function.

This higher level of connectivity has increased the attack surface of everyday objects, requiring security measures not previously necessary. In our example, a modern car contains more than fifty embedded computers, communicating through a very simple broadcast network called CAN bus, which offers no security features. However, introducing security protocols in the CAN bus requires to work within safety and cost constraints that are not present when designing security protocols for traditional computer systems.

To maintain the system schedulable and deliver signals in the car within strict deadlines, one cannot use challenge-response patterns, instead tracking the state of the system explicitly is required. The ability of tracking state is out of range for many of the most advanced analysis techniques, which obtain their efficiency by coalescing information about multiple states. However, we maintain that the same technologies can be extended to cope with this new challenge. Therefore we claim that

language based technologies offer a framework to push the boundaries of protocol verification, both in the symbolic and computational models, so as to encompass the verification of features peculiar to embedded systems.

This thesis sets to validate our claim by studying in depth a real-world use case in

automotive and extending state-of-the-art techniques as a result of our discoveries.

1.2 Contributions

To capture the peculiar features of the protocols under our consideration, we introduce Set-Pi, an extension of the Applied Pi-calculus with support of global non-monotonic state. Set-Pi adds sets and set transformations (insertion, deletion) as first class citizens to the Applied Pi-calculus.

The extension allows us to model succinctly the stateful aspects of our case study, and are general enough to express interesting properties of a wide range of stateful protocols, not limited to embedded systems: for example protocols with key revocation/update, protocols with databases etc.

We propose a new formulation of weak and strong agreement properties in terms of sets and transitions, which can contribute to a better understanding of existing formulations. Similarly to previous work in the Applied Pi-calculus [Bla02], we provide an analysis for Set-Pi that translates the process calculus models into logic programs, and either proves the absence of attacks in the model, or shows a potential attack. The analysis is implemented in a tool and successfully tested against a number of different case studies.

In order to build verified protocol implementations, we present a methodology for proving strong agreement properties in the computational model, using refinement types — types with attached formulas — to express such properties on the code. This method differs from previous work in that standard first order logic is used, hence protocols can be verified with existing tools which support first order logic and refinement types, like F*.

We analyse two security protocols in automotive, MaCAN and CANAuth, using both the symbolic and computational models. We adapt a standard schedulability analysis to take the security features into account, and to ensure that the protocols respect the safety constraints. Our analysis using the Applied Pi-calculus discovers a number of flaws in the MaCAN protocol and limitations in expressing the properties of our interest for both MaCAN and CANAuth. Throughout the thesis we present corrected models using our language Set-Pi, as well as a minimal verified implementation of CANAuth in F*.

The results of this study validate the utility of our formal approach, which helped us to discover and correct multiple flaws in the protocol design of MaCAN and to ensure that our modified designs are secure w.r.t. the original specification.

1.3 Structure of this Thesis

This thesis proceeds as follows:

- Chapter 2 presents technical background material on CAN bus and related technologies, including the security protocols MaCAN and CANAuth. These are later used throughout this thesis as the subject of our analysis.
- Chapter 3 introduces protocol verification in the symbolic Dolev-Yao model and the computational model. The techniques that form the basis for our contribution in both settings are presented as well, namely the Applied Pi-calculus/ProVerif and F*.
- Chapter 4 presents our case studies. We model them in the Applied Pi-calculus, finding flaws in one of the protocols, and showing the limitations that we get in the analysis. Considerations around security and schedulability will introduce us to Chapter 5. This work is based on [BSNN14] and the SESAMO deliverable [Pro13].
- Chapter 5 shows how to extend the standard schedulability analysis to take into account the security features of the protocol presented in Chapter 4. We also discuss how we found a DoS attack that arises in MaCAN, and how to eliminate it with acceptable costs.
- Chapter 6 presents Set-Pi, an extension of the Applied Pi-calculus which allows us to precisely capture the spurious attacks discovered in Chapter 4. The calculus can also be used in verifying protocols where non-monotonic sets of values (e.g. keys) are used. This chapter is based on [BMNN15].
- Chapter 7 presents an analysis of different authentication mechanisms in F*, a ML-like programming language with dependent types — types with attached formulas to data — which directly allows to prove properties on the programs. In this case we use it to prove authentication properties on a minimal implementation of the protocol. This chapter has been presented at the CryptoForma workshop [BKA+15].
- In Chapter 8 we present our final considerations, discuss related work and potential future work.
- In the appendix we present proofs of correctness for the analysis in Chapter 6.

CHAPTER 2

The CAN Bus Protocols

In this chapter we provide the necessary background for Chapter 4 and Chapter 5. We introduce relevant details about the CAN bus protocol, its extensions and the authenticated protocols MaCAN and CANAuth, that run on top of it.

2.1 Background

During the '80s car manufacturers developed the first vehicles that included Electronic Control Units (ECUs), often employing custom solutions for implementing communication between them, or resorting to costly off-the-shelf solutions that were not fit for the newly born and growing automotive sector. There was a need for a standardised, cost efficient, and reliable protocol that could fit the requirements of the automotive domain.

In 1983 Robert Bosch GmbH started working on a prioritised real-time bus network protocol for resource limited microcontrollers that could fit the precise needs of automotive applications. In 1986 Bosch officially released their Controller Area Network (CAN) bus protocol, and in 1987 Intel and Philips released the first two microcontrollers with hardware support for this protocol.

Since then, CAN became the primary choice for car manufacturers, and its application areas expanded to big vehicles like trucks and agricultural machines, medical equipment

and other forms of automation. The CAN protocol was standardised in 1993 as ISO-11898, and later revised in 1995. It is also required for on-board diagnostics of vehicles sold in the United States since 1996, as mandated by the OBD-II standard, and for vehicles sold in Europe since 2001, mandated by the EOBD standard.

The limitations of the microcontrollers that were available in 1983 are reflected in the protocol design, which presents simple features — broadcast, fixed message priority, unauthenticated — and fault tolerance mechanisms, namely error detection with CRC codes and automatic disconnection of faulty nodes by the use of error flags.

Technology evolved over time, and more and more ECUs have been introduced in modern vehicles. Some have been used for reducing production costs and maximise efficiency, like the engine control unit. Others increased the vehicle safety, like the Anti Blocking System (ABS) which avoids uncontrolled skidding and increases traction during braking. Again others have been introduced to add new features to a car: among these, the car infotainment system and the telematics unit can now provide navigation information to the user, manage vehicle functionality to ease the driving experience (e.g. automatically managing music volume, seat position given user preferences, etc.) or add anti-theft mechanisms that integrate GPS information and proprietary protocols over GSM communication for tracking and blocking stolen vehicles.

These new functionalities result into an increased level of connectivity inside the car and higher complexity of the units responsible for controlling the vehicle. Figure 2.1 summarises the network topology and connectivity of a modern vehicle, showing a low speed/low criticality CAN bus network and a high speed/highly critical CAN bus network, connected to the telematics unit and the infotainment system. These two act as an interface to the external networks. Modern vehicles have several wireless and wired interfaces, including WiFi, Bluetooth, remote monitoring/anti-theft protocols over cellular network, diagnostic interfaces, GPS signaling system, and wireless keys.

In two recent studies [KCR+10, CMK+11] a joint group of researchers from University of California San Diego and the University of Washington analysed the vulnerability of a modern sedan. They were able to reverse-engineer their behaviour, taking apart the control units. In their first work [KCR+10] they managed to remotely control the vehicle from the U.S. mandated OBD-II diagnostic port, which directly communicates over the low speed network to the telematics unit. Through bad filtering and broadcasting policies they were able to send message from the low-speed network through the high speed network of the car, achieving complete control of the vehicle.

Later [CMK+11] they attacked the remote vulnerabilities of the vehicle and found an array of potential security flaws: a buffer overflow in the Bluetooth stack of the infotainment system, and another buffer overflow in the WMA codec of the media player, allowed them to create a CD that was capable of running the same attack of their previous experiment.

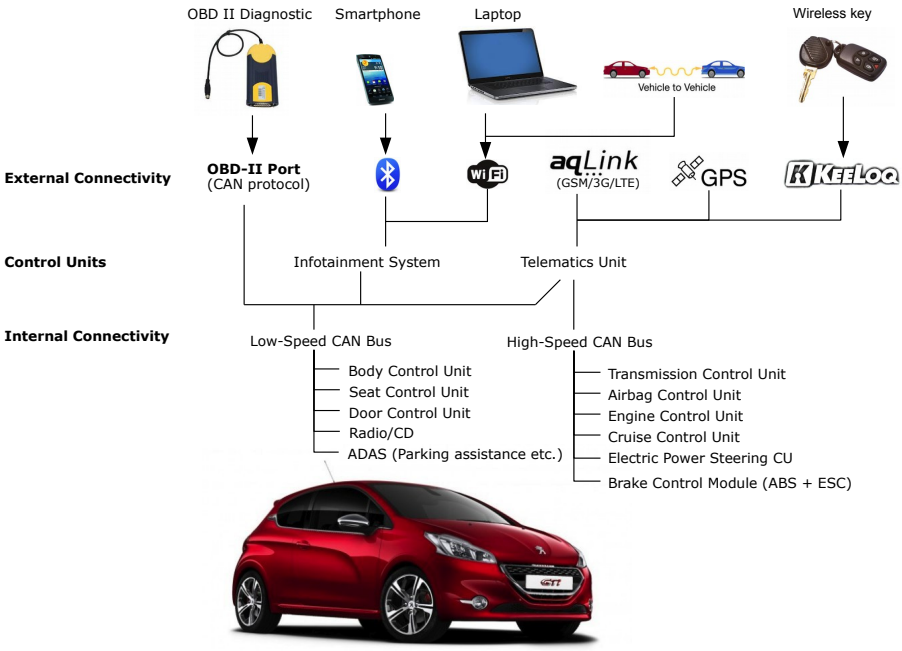


Figure 2.1: Connectivity inside a modern vehicle

The unauthenticated and broadcast nature of CAN makes it hard to implement any kind of information flow control, and even some basic filters required by the safety standard were not properly implemented due to diagnostic requirements [KCR+10], leading to leakage of signals from the low-speed network to the high-speed network. These first attacks have sparked an enormous interest across the academic, industrial and hacking communities, and new exploits for different models of vehicles continuously hit the news.

Among the more interesting recent exploits we mention the work of Miller and Valasek [MV13, MV14, MV15]. They published a series of Black Hat security papers studying different makes and models of vehicles finding various security vulnerabilities. The last of the series [MV15] shows the first discovered remote exploit on a vehicle, proving that these attacks are possible in practice.

Another interesting document, freely available, is the "Car Hacker's Handbook" [Smi14]. This document teaches how to hack a car to anyone who has basic hardware and software skills and a few widely available tools: hacking a car has never been easier. One of the root causes for the weakness of current automotive architectures lies in the design of the CAN bus, which was conceived as a simple, economic and reliable protocol, but with no security in mind.

To mitigate the problem, different groups have proposed authenticated extensions to the CAN protocol [HSV11, GMHV12, GGH+12, HRS12, SRW+11]. These authenticated extensions add a signature for proving the origin of the message, its intended recipient, and for ensuring freshness. They are necessary to enforce information flow policies such as "only the brake pedal and the anti-blocking system are allowed to activate the brakes" or "messages coming from a diagnostic unit connected through the OBD-II port can only be accepted in diagnostic mode", and to prevent replay attacks of previously sent messages.

In the next two sections we are going to give an overview of CAN and related technologies, then introduce two authenticated protocols on top of CAN — MaCAN and CANAuth — that will be the subjects of our studies throughout the thesis.

2.2 CAN Bus and Extensions

2.2.1 CAN Bus

CAN is a broadcast, prioritised, real-time protocol designed to work on a bus network topology of interconnected microcontrollers. Typical transmission speeds of the CAN

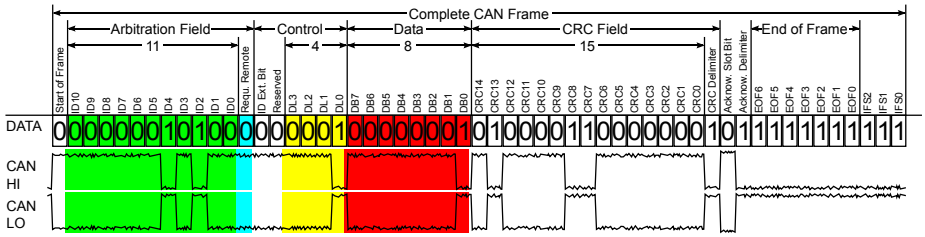


Figure 2.2: A typical CAN frame[1]

bus are 125 Kbps, 500 Kbps and 1 Mbps.

As shown in Figure 2.2, a CAN frame is composed by an ID (or arbitration) field of 11 or 29 bits (also used for arbitration), 6 control bits, including 4 defining the payload length (in bytes), a payload of 0 to 64 bits, a 15 bit CRC code, 2 bits for acknowledgement, and a sequence of 7 bits delimiting the end-of-frame.

The ID/arbitration field of a CAN frame is used for priority scheduling, where lower ID values have higher priority. All ECUs that are waiting to send a message on the CAN bus start transmitting the message IDs during the arbitration phase, using a Carrier Sense Multiple Access/Bitwise Arbitration (CSMA/BA) scheme. In this scheme each ECU sends its message ID bit by bit, while sensing on the channel if other ECUs are also transmitting. When an ECU sends a 1 bit of ID while reading a 0 bit on the channel, it stops communicating and waits for the next frame, giving priority to the other ECU, which has a lower message ID.

This arbitration scheme produces a simple and predictable system, that can be analysed to give guarantees on the real-time schedulability of a configured network, as we will see in Chapter 5.

Given the restricted size of CAN frames, all extensions that add authentication to the network must make a compromise on the space-time-complexity constraints. As we will see later, both CANAuth and MaCAN use weak but fast cryptography. Both protocols rely on extensions of the CAN Bus that allow longer payloads to avoid sacrificing the standard message length, or worse splitting frames, which would likely interfere with the real-time schedulability constraints of the applications. We now briefly present the extensions.

2.2.2 CAN+ and CAN-FD

CAN+ [ZWT09] is an extension of the CAN bus that makes it possible to encode additional bits of payload in a CAN frame, while remaining backwards compatible. This is done by exploiting the fact that CAN devices sample a bit signal at approximately 75% of the time frame allocated to a bit transmission, in order to wait for the channel to stabilise. With new and more precise equipment, it is possible to encode more bits between 15% and 55% of the time frame, hence encoding at least 15 additional bits for every bit of payload transmitted on a 1Mbps CAN bus, and possibly even more in case of slower networks.

One particular advantage of this approach is that the added payload is completely transparent to standard CAN devices. CAN+ devices can be integrated in a CAN network and — if the authenticated extensions are restricted on the additional bits of payload — they can easily be ignored by non-CAN+ enabled ECUs. On the other hand CAN+ requires more expensive equipment, which could deter from to its adoption.

CAN-FD [Har12] allows transmitting up to 64 bytes of payload by using a Flexible Datarate (FD). The message arbitration phase proceeds as in standard CAN at the normal data rate, while the payload and CRC codes are sent at a faster data rate, allowing extended payload length. The payload length field has a different interpretation in CAN-FD (i.e. 1), so that the timing remains compatible with standard CAN devices.

CAN bus hardware can also be used for CAN-FD, making it a much more attractive evolution of the current car infrastructure. Differently from CAN+, the extended frames in CAN-FD cannot be interpreted by standard CAN controllers, so this extension could not support transparent authentication. Bosch is currently pushing for its adoption, so it is very likely that it will be the standard of choice for adding authentication features to this very simple protocol.

2.2.3 TTCAN

TTCAN (Time Triggered CAN) is an extension on top of the standard CAN protocol that allows both time triggered and event triggered communication. The scheduling of time triggered communication is defined statically at development time, while event triggered communication (the standard mode in CAN) is allowed to happen only during defined arbitration windows, and follows the standard priority based non-preemptive scheduling mode.

The specific features of TTCAN result into increased real-time performance, and easier

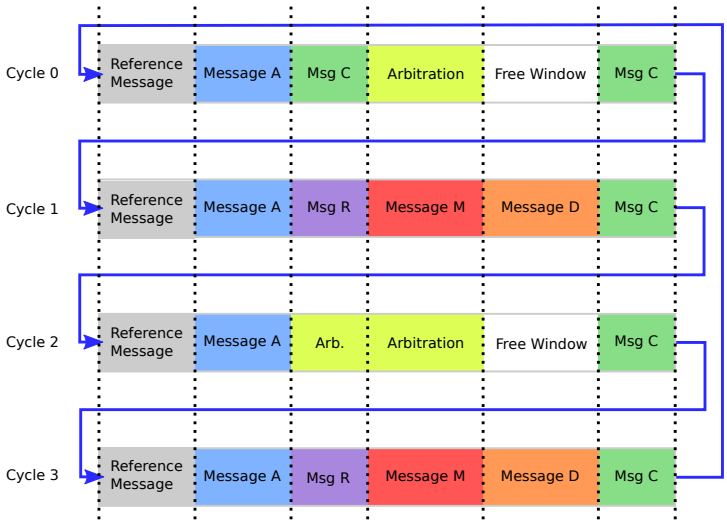


Figure 2.3: Example of TTCAN scheduling, showing four basic cycles, each starting with the reference message, three arbitration windows in cycles 0 and 2, two free windows in cycles 0 and 2, and the statically scheduled messages in the remaining slots.

schedulability for periodic messages: it is no longer needed to compute a worst-case response time analysis for periodic messages, since they have fixed scheduling and therefore fixed response times. Lastly, since TTCAN allows both time triggered and event triggered communication in the same network, there is no need for duplicating the hardware to support both modes, hence reducing implementation costs.

In TTCAN the communication is organised in *basic cycles*, predefined time frames with a fixed combination of *exclusive*, *free* and *arbitrating* windows. At the start of each basic cycle, a chosen master ECU sends a *reference message*, that provides information for time triggered control and allows all other ECUs to synchronise. *Exclusive windows* are static time frames where only one predefined ECU at a specific time is allowed to communicate over the network, *arbitrating windows* are dedicated to standard CAN arbitration for event-triggered messages, while *free windows* are empty slots dedicated to extensions, like the integration of third party accessories in a vehicle. A sequence of basic cycles is called *system matrix*, which is continuously repeated over the execution of the protocol. Figure 2.3[2] shows an example TTCAN system matrix, graphically representing all the concepts just explained. For a detailed description of how communication is handled in TTCAN we refer the reader to [FMD+00, HMF+02].

[2]Source: `http://www.can-cia.org/index.php?id=166`

2.3 Authenticated extensions

In this section we discuss the desired *properties* that authentication protocols in automotive should enforce, and the specific *constraints* in which they have to operate. These considerations led us to choice of studying MaCAN and CANAuth among a plethora of proposals. In fact, these were the only protocols that met all the requirements. We will present them in the next two subsections.

Properties We identified the following set of properties that authenticated protocols in safety critical embedded systems should enforce. We relate these properties back to Lowe's taxonomy for authentication [Low97].

Key secrecy During the execution of the protocol the keys that are exchanged don't get revealed to the attacker.

Authentication If A is sending a message and B acknowledges it, then it is really A who sent the message. The protocols we are going to analyse use keyed hashing during session communication, so in principle any participant with the valid key to check a message is also able to sign it. This results into a weaker notion of authentication: if a message with a valid signature is sent over the network then it is signed by one of the trusted parties. Nevertheless this can be related back to Lowe's notion of *weak (non-injective) agreement*, or weak authentication, since all participants in a group can be viewed as a single principal.

Freshness If an authenticated message is sent through the network then the same content and signature cannot be accepted twice. Freshness violations are also called *replay attacks*. This notion of freshness correspond to the concept of *(injective) agreement* in Lowe's taxonomy.

Schedulability The real-time system must be schedulable (i.e. deliver the messages within the deadlines) even under the presence of an attacker. We consider this equivalent to the notion of *recent agreement* in Lowe's taxonomy. Since it is not possible to guarantee this in principle, we study which assumptions need to be placed in the attacker model to provide such a guarantee.

Constraints The real-time nature of the CAN protocol imposes tight constraints on a potential extension: first of all, authentication should be compatible with the *timing constraints*. Since the current usage of the network is at about 80% of its capacity in current applications [DBBL07], one cannot choose to use more traditional schemes like challenge-response protocols, instead authentication has to fit into one single CAN message.

Since the protocol is running on microcontrollers with limited processing power, the cost of computing the cryptographic primitives must be limited in order to respect the deadlines imposed on the system. The response time for message transmission is clearly affected by the processing time required by the sender to sign the message and by the receiver to check the signature.

The second source of constraints is the *space limitations* that arise from the limited payload length offered by CAN. Since authentication must take place in a single frame transmission, it must either occupy part of the payload, limiting its size and the strength of the cryptographic primitive as well, or use one of the two extensions (CAN+, CAN-FD) above presented.

2.3.1 CANAuth

CANAuth [HSV11] is a compatible extension to the CAN protocol that adds authentication features to the standard. It accomplishes this goal by exploiting the CAN+ extension of the CAN bus, which allows to add information by encoding extra bits in between the sampling points of a traditional CAN transmission [ZWT09].

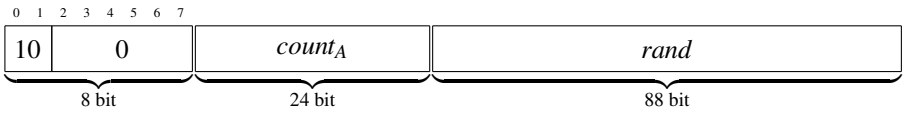
CANAuth provides a mechanism to verify message authenticity (but not the origin due to limits in the CAN protocol), resistance to replay attacks, and group keys management functionality while maintaining backwards compatibility with the CAN standard. Multiple ECUs can share a single authentication key for a group of messages. At the low level, patterns and masks on the message identifiers define to which authentication group a message belongs to.

CANAuth uses the CAN+ extension to provide authenticated messages that are compatible with standard CAN ECUs. Since 120 bits is the lower bound of additional information that can be used per each CAN packet[3], CANAuth accomplishes its goal within this space constraint.

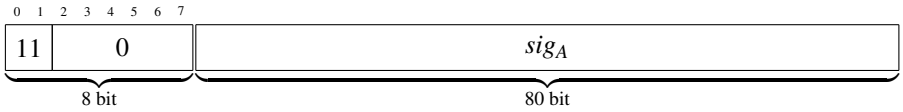
The protocol consists of two phases: the first is *key establishment*, where a designated master ECU initiates authenticated communication by establishing a session key that will be used to authenticate messages; the second is *message authentication*, where a message sent through the channel is signed with the session key previously communicated.

Key Establishment All ECUs connected to the CAN network have at least one pre-shared key kp installed. To establish a session key (2.1) the designated master ECU_i

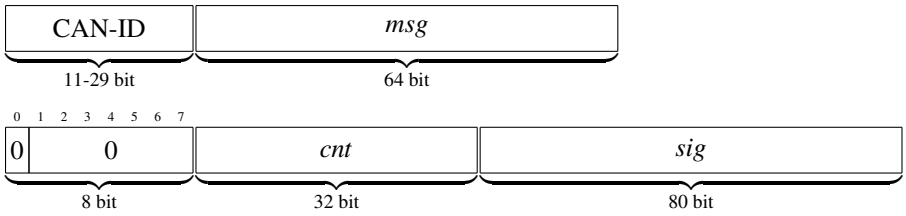
[3] This happens with a 1 MHz CAN bus, as the one used in the network that runs high-integrity ASIL C-D components.



(a) CANAuth frame for key establishment, containing the counter and random numbers.



(b) CANAuth frame for key establishment, containing the signature.



(c) CANAuth frame for message authentication. The first row represents the standard CAN ID and payload fields, while the second row represents the CAN+ extension payload.

Figure 2.4: CANAuth message authentication frame

broadcasts a counter (*countA*, 24 bits) and a random number (*rand*, 88 bits). The counter must be greater than every value already used during key establishment in order to ensure freshness. At this stage every ECU in possession of the pre-shared key can compute the session key (2.2) and the signature (2.3) using the received information. To confirm that the transmission succeeded, the master ECU again sends the signature (2.4), so that the other nodes in the network can compare it with their own computed value.

$$ECU_i \rightarrow ECU_j : \mathsf{CH}, count_A, rand \tag{2.1}$$

$$ks = hmac(kp, \langle count_A, rand \rangle) \pmod{2^{128}} \tag{2.2}$$

$$sig_A = hmac(ks, \langle count_A, rand \rangle) \pmod{2^{112}} \tag{2.3}$$

$$ECU_i \rightarrow ECU_j : \mathsf{SIGN}, sig_A \tag{2.4}$$

The session key *ks* is 128 bit long and relies on the strength of the hashing function HMAC [KCB97]. The modulo operation is not required for the abstract protocol as HMAC can produce longer signatures and *ks* is never sent through the network. However it is necessary to fit the message in one frame, increasing the speed of the authentication

process.

Message Authentication Once a session key is established, messages can be authenticated through the channel. The message format, shown in Figure 2.4, shows the sizes of the bit fields, where the first row represents the CAN bus frame (with 64 bit of payload) and the second row represents the CAN+ extension (120 bit). To authenticate a message M, ECU_i sends a counter ($count_M$, 32 bits) and the signature sig_M in (2.6). To ensure freshness $count_M$ has to be greater than any other previously used value.

$$sig_M = hmac(ks, \langle count_M, M \rangle) \pmod{2^{80}} \tag{2.5}$$

$$ECU_i \rightarrow ECU_j : M, \mathsf{SIGN}, count_M, sig_M \tag{2.6}$$

2.3.2 MaCAN

MaCAN [HRS12] is an alternative proposal for doing authentication over the CAN bus. The authors argued that there was a need for a different authentication scheme than the one proposed by CANAuth. CANAuth relies on the extra bits offered by CAN+, and therefore requires also switching to different hardware. Furthermore, it is more likely that CAN-FD is going to be adopted.

MaCAN is a protocol that provides authentication capabilities while meeting the real-time requirements of the traditional CAN networks. It is designed to be flexible and backwards compatible, so as to accommodate a mixed environment of CAN and MaCAN ECUs.

Single message authentication is done in 32 bits using the CMAC [Dwo05] keyed hashing function, that fits in half of the standard CAN payload. The signature length can also be extended when using CAN-FD in order to increase robustness and to allow more than 64 bits of payload.

The designers of MaCAN discarded the use of more traditional challenge-response protocols during the second phase of message authentication, because of the real-time requirements that need to be met in the applications. They also considered problematic the use of counter values to ensure message freshness. This is justified by the fact that some ECUs in the automotive environment are unavailable at times due to sleep cycles, thus creating synchronisation issues [HRS12].

The Crypt Frame As depicted in Figure 2.5 the crypt frame is a specific interpretation of the traditional CAN frame where the traditional CAN fields are split to encode

authentication details. This format constitutes the basis for MaCAN's additional functionality.

The CAN ID in the crypt frame contains a source ID of 6 bits, which indicates the source ECU. The first byte of data is used to send a 2 bit flag field and a 6 bit destination id, which could indicate a specific ECU or a group of ECUs, in order to make the protocol fully directional.

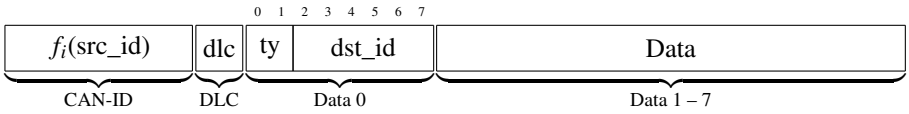


Figure 2.5: The MaCAN crypt frame

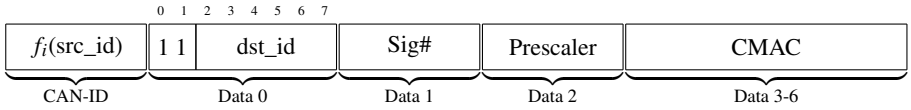
Key establishment The key establishment phase of the protocol works as follows. ECU_i sends a challenge C_i to the designated Key Server (KS), signalling the requested partner ECU_j (2.7). KS replies with the session key $SK_{i,j}$ encrypted, together with the challenge and the two identifiers of the two participants, with the pre-shared key $K_{i,ks}$ (2.8). KS also sends a challenge request to ECU_j, in order to activate it (2.9).

After ECU_i decrypts the session key, it sends an acknowledgement to the other partner, authenticating it with a CMAC of its ID, $group\_field$ and a timestamp in order to ensure freshness (2.10). $group\_field$ is a 24 bit field that signals which ECUs in a particular group are already authenticated according to ECU_i. After sending this message, ECU_i considers itself authenticated to the group.

Then a challenge is also sent from ECU_j to KS and the protocol continues symmetrically in order to authenticate also ECU_j to ECU_i (2.11–2.14).

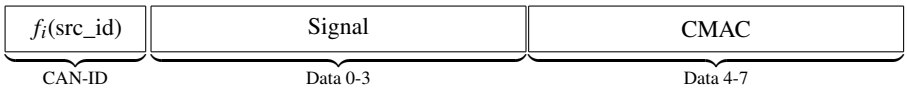
$$ECU_i \rightarrow KS : \mathsf{CH}, C_i, id_j \tag{2.7}$$
$$KS \rightarrow ECU_i : \mathsf{SK}, senc(K_{i,ks}, C_i, id_j, id_i, SK_{i,j}) \tag{2.8}$$
$$KS \rightarrow ECU_j : \mathsf{RC} \tag{2.9}$$
$$ECU_i \rightarrow ECU_j : \mathsf{ACK}, group\_field, cmac(SK_{i,j}, T, id_j, group\_field) \tag{2.10}$$
$$ECU_j \rightarrow KS : \mathsf{CH}, C_j, id_i \tag{2.11}$$
$$KS \rightarrow ECU_j : \mathsf{SK}, senc(K_{j,ks}, C_j, id_i, id_j, SK_{i,j}) \tag{2.12}$$
$$ECU_j \rightarrow ECU_i : \mathsf{ACK}, group\_field, cmac(SK_{j,i}, T, id_j, group\_field) \tag{2.13}$$
$$ECU_i \rightarrow ECU_j : \mathsf{ACK}, group\_field, cmac(SK_{i,j}, T, id_j, group\_field) \tag{2.14}$$

Message authentication Messages in MaCAN can get authenticated in two different modes (and formats): the first mode is used to authenticate only the successive signal, and allows only 16 bits of payload in the standard configuration. The second mode is used to authenticate every broadcast signal that is sent, and allows 32 bits of payload using CAN (the other 32 bits are taken by the CMAC signature). This is done by setting the *Prescaler* to either 0 for having the next message authenticated, or $n \geq 1$ for having every n-th message authenticated. Figure 2.6 shows the frame layout for each message in the narration.



(a) Signal authentication request frame (SIG_AUTH_REQUEST, 2.15)

(b) Crypt frame with 32 bit signature (SIG_AUTH, 2.16)



(c) Standard CAN frame with 32 bit signature (SIG_AUTH, 2.17)

Figure 2.6: Frame formats for authentication

The protocol proceeds as follows: in (2.15) ECU_i sends a signal authentication request to ECU_j, declaring which signal needs to be authenticated (*Sig#*), a *Prescaler* that specifies the frequency of authenticated signals, and a signature for the message authenticity. Depending on the choices made at design time, ECU_j responds in one of the two formats, sending the signal and the signature of the signal concatenated with the communicating parties and a timestamp to ensure freshness (2.16–2.17).

$$ECU_i \rightarrow ECU_j : SIG\_AUTH\_REQ, Sig\#, Prescaler, \\ cmac(SK_{i,j}, \langle Time, id_i, id_j, Sig\#, Prescaler \rangle) \tag{2.15}$$

$$ECU_j \rightarrow ECU_i : SIG\_AUTH, Sig\#, Signal, \\ cmac(SK_{i,j}, \langle Time, id_i, id_j, Signal \rangle) \tag{2.16}$$

$$ECU_j \rightarrow ECU_i : SIG\_AUTH, Signal, \\ cmac(SK_{i,j}, \langle Time, id_i, id_j, Signal \rangle) \tag{2.17}$$

Serving time All signatures include a timestamp that is not sent in cleartext over the channel, thus the communicating ECUs need to be perfectly synchronised. If an ECU is not synchronised, it has no information about the current timestamp, therefore it is not able to check the signature and recognise a message as valid. This factor produces synchronisation problems between ECUs, and thus a time server is introduced to globally provide the necessary time information.

When a ECU (ECU_i) needs to synchronise its internal clock with the global time, it sends a challenge to the time server (2.18), and this later responds with the current time information, signed with the challenge just received and the key shared only between $SK_{ts,i}$ (2.19).

$$ECU_i \rightarrow TS : \mathsf{CH}, C_i, fwd\_id = 0 \tag{2.18}$$

$$TS \rightarrow ECU_i : Time_{t-1}, cmac(SK_{ts,i}, \langle C_i, Time_{t-1} \rangle) \tag{2.19}$$

2.4 Conclusions

Introducing security protocols in embedded systems — which were not initially designed with security in mind and must obey to safety and cost considerations — presents tough design choices that would not be required in a more traditional setting. Both MaCAN and CANAuth make concrete choices, such as the length of signatures and keys, which would be risible if considered out of context, but considering the typical usage of a car and the bandwidth limitations of the internal network provide a sufficient assurance within the tight constraints.

Having presented the concrete scenario, we now proceed to Chapter 3 by introducing the necessary theoretical background on which we base our contribution.

CHAPTER 3

Formal Protocol Verification

3.1 Introduction

Formal protocol verification aims at verifying security protocols by the use of formal methods. Verification of security protocols is an area where formal methods have flourished in the last two decades. The goal is to find, given an attacker model, a protocol description and the desired security properties, whether the protocol is secure with respect to the attacker, or the security properties are violated.

As opposed to traditional security proofs, which protocol designers and cryptographers carry out by hand and are thus error prone, formal methods in security verification aim at fully machine checked proofs, and possibly automated ones. Both in traditional security proofs and in (semi-)automated verification, there have been two basic models for protocol verification: the *symbolic model* — often referred to as Dolev-Yao — and the *computational model*.

Briefly speaking, the symbolic model assumes that cryptographic functions are perfect (i.e., unbreakable), and encodes them as operations on algebraic terms. By contrast, the computational model assumes makes finer assumptions on the cryptographic functions. In the computational model, cryptographic functions are programs that operate on

bitstrings, and the security argument relies on assuming that breaking the security mechanisms is a computationally hard problem.

In this chapter we are going to present verification techniques in both models. As we will see, they both provide a different combination of advantages and shortcomings in their application to formal methods.

3.2 Symbolic Model

The Dolev-Yao model [DY83] is a well-known attacker model for analysing security protocols. Dolev-Yao assumes that the cryptographic primitives are perfect — hence cannot be reversed without knowing the keys. It also assumes that the attacker knows all public information, can freely manipulate the public channels, encrypt and decrypt messages, and generate fresh values. Assuming that the cryptographic primitives are perfect allows modelling protocols in symbolic terms, which greatly simplifies protocol analysis.

For example, the symmetric encryption of a plaintext p with a key k in the Dolev-Yao model is represented with the term $senc(k,p)$, and the corresponding decryption function of ciphertext c is represented with the term $sdec(k,c)$, and the following equation holds:

$$sdec(k,senc(k,p)) = p \tag{3.1}$$

In reality, symmetric encryption is supposed to be *hard* to invert, and it is possible to recover the ciphertext from the plaintext with a certain amount of work. However, (3.1) matches the perfect cryptography assumption in the symbolic model, where an encryption function cannot be inverted unless the key is available.

As another example, hash functions are considered to be injective in the symbolic model, hence we assume $h(m) = h(m') \implies m = m'$. Real hash function implementations cannot enjoy this property, because their codomain is strictly smaller than their domain. However, cryptographically secure hashing functions have a good probability distribution in their codomain, and finding two messages m_1 and m_2 for which $h(m_1) = h(m_2)$ is supposed to be a hard problem. Figure 3.1 summarises how to encode a number of common cryptographic primitives.

One particularly successful symbolic approach is to represent security protocols with Horn clauses, which can then be solved using any Prolog system. In the next section we detail how one can encode protocols into Horn clauses. In Section 3.3 we present the Applied Pi-calculus [AF01], a process calculus designed to describe security protocols in the symbolic model, and we show how to systematically analyse Applied Pi-calculus programs by translating them into Horn clauses.

	Function	Inverse
Symmetric encryption:	$senc(k,m)$	$sdec(k,senc(k,m)) = m$
Asymmetric encryption:	$aenc(pk(x),m)$	$adec(sk(x),aenc(pk(x),m)) = m$
Signatures:	$sign(k,m)$	—

Figure 3.1: Symbolic encoding of cryptographic primitives.

$$
\begin{aligned}
M &::= x \mid a[M_1,\ldots,M_n] \mid f(M_1,\ldots,M_n) && \text{variables, names and functions}\\
P &::= \mathsf{p}(M_1,\ldots,M_n) && \text{predicates}\\
\mathscr{C} &::= P_1 \wedge \cdots \wedge P_n \implies P && \text{Horn clauses}
\end{aligned}
$$

Figure 3.2: Syntax for Horn clauses

3.2.1 Horn Clause Representation

The idea of representing security protocols with Horn clauses was initially conceived by Weidenbach [Wei99] and later used in ProVerif [BS13]. ProVerif introduced a specific resolution procedure for Horn clauses, that has shown experimentally better termination and performance results than other Prolog systems [Bla01].

ProVerif has also the ability to verify injective and non-injective agreements, by adding events to the Applied Pi-calculus and extending the analysis to verify non-injective and injective agreements, according to Lowe's [Low97] definitions. For a detailed presentation of the Applied Pi-calculus with events and the analysis techniques used in this thesis we refer to [Bla09].

Before discussing how to represent security protocols by means of Horn clauses, we introduce in Figure 3.2 the syntax that will be used throughout the thesis. M ranges over terms, P represents predicates and $\mathscr{C}$ represents clauses. We differ from standard logic syntax in that we distinguish algebraic constructors into names ($a[\cdot]$) and functions ($f(\cdot)$). We use the name notation to identify values like nonces and keys, and the function notation to denote cryptographic primitives.

Representing security protocols Hereby we give an informal description of the most common encoding of security protocols into Horn clauses. Section 3.3 will show how to systematically construct Horn clauses, starting from protocol models described in the Applied Pi-calculus.

Horn clauses represent the interactions of a protocol with an attacker on a public channel, and model the behaviour of the protocol and the attacker capabilities. They are better explained with an example. Consider the well-known Needham-Schröder protocol, where the two principals A and B want to establish a secure connection over an insecure network using a trusted server S:

$$A \to S : A, B, N_A \tag{3.2}$$
$$S \to A : \{N_A, K_{AB}, B, \{K_{AB}, A\}_{K_{BS}}\}_{K_{AS}} \tag{3.3}$$
$$A \to B : \{K_{AB}, A\}_{K_{BS}} \tag{3.4}$$
$$B \to A : \{N_B\}_{K_{AB}} \tag{3.5}$$

The initiator A contacts the server indicating that it wants to communicate with B, and uses a fresh nonce N_A to mark a new session (3.2). In response S returns an encrypted message with the key K_{AS} pre-distributed to A and S. The message contains the nonce N_A, a key for the session K_{AB}, the ID of the responder B, and a message for B containing the current session key K_{AB} and the ID of the initiator A, encrypted with the shared key K_{BS} between B and S (3.3). The initiator is then able to decrypt such message and extract the encryption intended to B (3.4), who finally receives the session key K_{AB} and can start communicating with A (3.5).

The protocol can be encoded with the following Horn clauses:

$$\Longrightarrow \mathsf{att}(a[], b[], n_A[]) \tag{3.6}$$
$$\mathsf{att}(a[], b[], n_A[]) \Longrightarrow \mathsf{att}(senc(k_{AS}[], \langle n_A[], k_{AB}[], b[], senc(k_{BS}[], \langle k_{AB}[], a[]\rangle)\rangle)) \tag{3.7}$$
$$\mathsf{att}(senc(k_{AS}[], \langle n_A[], k_{AB}[], b[], senc(k_{BS}[], \langle k_{AB}[], a[]\rangle)\rangle)) \Longrightarrow \mathsf{att}(senc(k_{BS}[], \langle k_{AB}[], a[]\rangle)) \tag{3.8}$$
$$\mathsf{att}(senc(k_{BS}[], \langle k_{AB}[], a[]\rangle)) \Longrightarrow \mathsf{att}(senc(k_{AB}[], n_B[])) \tag{3.9}$$

Clause 3.6 represents the message 3.2 sent from A to the server, stating that it wants to communicate with B using the fresh nonce n_A: $\mathsf{att}(a[], b[], n_A[])$ denotes that such message is sent through the public channel and hence the attacker knows it. Clause 3.7 represents message 3.3, sent from the server to A. The message 3.2 is required to activate the server, and therefore it is marked as a hypothesis to the clause. Similarly, the clauses 3.8 and 3.9 represent the messages sent by A and B in the communication steps 3.4 and 3.5, respectively.

These clauses denote the interaction between the principals in the protocol, where each clause corresponds to an output. Each party in the communication is receiving and

sending messages: in this encoding each input becomes a *precondition* to a clause, and each output becomes a *conclusion* (or head) for the clause. The predicate $\mathtt{att}(\cdot)$ represents the values observed by the attacker over the public channel, where the argument of the predicate is a term in the Dolev-Yao model.

Cryptographic primitives are also represented by Horn clauses. In the Herbrand universe $h(m) = h(m') \Leftrightarrow m = m'$ holds, which as we saw models hash functions. The equation $sdec(k, senc(k,m)) = m$ represents the capability of the attacker to decrypt a message, hence it can be encoded as:

$$\mathtt{att}(senc(k,m)) \wedge \mathtt{att}(k) \implies \mathtt{att}(m) \tag{3.10}$$

that is, if the attacker knows an encrypted term, and it knows the key that was used for the encryption, then it is able to derive the plaintext.

A secrecy goal can be represented as a query of the form $\mathtt{att}(x)$ where x is the term that has to be proven secret. Let $\mathscr{C}_P$ be the set of clauses that describe the behaviour of the protocol and the capabilities of the attacker. If the formula $\mathscr{C}_P \wedge \neg\mathtt{att}(x)$ is satisfiable, then the protocol is secure, otherwise there might be an attack.

3.2.2 Saturation

The property is checked by *saturating* the set of Horn clauses $\mathscr{C}_P$. Intuitively, in the saturation process, two clauses $H_1 \implies C_1$ and $H_2 \implies C_2$ are combined together if the head of the first clause C_1 "matches" with a fact in H_2; the resulting clause will have the hypotheses of both clauses, minus the hypothesis of H_2 that matches with C_1. The combined clause is inserted in the knowledge base, while all subsumed clauses that are present in the knowledge base are eliminated as redundant.

A specific goal (e.g. $\Rightarrow \mathtt{att}(x)$) is reachable if a clause exists in the saturation $sat(\mathscr{C}_P)$ that concludes $\mathtt{att}(x)$ and has no hypotheses. If such clause is found, then the proof tree for that derivation describes a potential attack, which can be reconstructed by the rules that are applied.

For example, if the rule (3.10) is used in a derivation for an attack, then the attacker has to perform a decryption; if the rule (3.6) is used, then the first message of the Needham-Schröder protocol is exchanged during the attack, and so on. If after an exhaustive search no derivation is found, then the model preserves the desired secrecy property.

An exhaustive search over the infinite space of derivable facts can easily lead to non-termination. Weidenbach's original work in encoding protocols into Horn clauses [Wei99] shows that the SPASS theorem prover does not always terminate in its search.

He also provided decidability results for the saturation of the particular logic fragment that he used. Blanchet later proposed a modified resolution procedure based on saturation that yields better results in practice [Bla01], implemented in ProVerif. It is proven to be sound and complete, semi-decidable procedure. The remainder of this section summarises this form of saturation, and Theorem 1 shows its correctness.

ProVerif's resolution procedure treats differently predicates that have the same form of a predicate in a set S. We define such predicates to be in S-form. The procedure consists of two phases: the first, which we will call S-saturation, takes an initial set of clauses B_0 (which we distinguish syntactically by the use of $\rightarrow$ instead of $\Longrightarrow$) and produces a new set of clauses B_1 where all the hypotheses are in S-form. The second phase tries to find a derivation for the particular query $\Rightarrow Q$ using the rules in B_1.

S-saturation The S-saturation phase consists in applying the following rule until no new clauses can be introduced in B_1.

$$\frac{H \rightarrow C \quad F_0 \wedge H' \rightarrow C'}{\sigma H \wedge \sigma H' \rightarrow \sigma C'} \text{ if } \begin{array}{l}(1)\ \sigma = mgu(F_0, C),\\ (2)\ F_0 \notin_r S, \text{ and}\\ (3)\ \forall\ F \in H,\ F \in_r S\end{array}$$

where the relation $F \in_r S$ intuitively means that the predicate F is of the same form of one of the predicates in S, and by default $S = \{\mathtt{att}(x)\}$.

Once a clause is produced by this rule, its variables are renamed apart so that they are fresh in B_1, and duplicate hypotheses are removed. The resulting clause is inserted in the knowledge base B_1, and all other clauses that it subsumes are removed from B_1.

The saturated set B_1 is produced by applying the saturation rule, removing duplicate hypotheses and subsumed clauses from the knowledge base, until a fix-point is reached. Finally, all the rules that do not satisfy (3) are removed from the knowledge base.

Backwards chaining The second phase is backward chaining, and is exactly as in standard Prolog. This procedure ends when it produces a clause that concludes the query goal Q and contains no hypotheses.

$$\frac{H \rightarrow C \quad F_0 \wedge H' \Rightarrow Q}{\sigma H \wedge \sigma H' \Rightarrow \sigma Q} \text{ if } \sigma = mgu(F_0, C)$$

THEOREM 1 *(Correctness of S-saturation) If Q is a closed fact (and S is chosen such that S-saturation terminates and backward chaining terminates on both B_0 and B_1),*

then
(a) $\Rightarrow Q$ *is obtainable by backward chaining using the given set clauses* B_0
iff
(b) $\Rightarrow Q$ *is obtainable by backward chaining using the S-saturated clause set* B_1.

PROOF.($b \implies a$) If $\Rightarrow Q$ is obtainable by the saturated rules in B_1, then it is also obtainable by rules in B_0.

Let T be the derivation tree for $\Rightarrow Q$ using rules in B_1. It is easy to see that a rule $R \in B_1$ is either already a rule in B_0 or it is constructed by the saturation process. If $R \in B_0$ and is used in T, then the derivation tree needs no change. If $R \notin B_0$, then we can modify the derivation tree in such a way that it only uses rules in B_0 in place of R.

Let T_R be the derivation tree for R using saturation. All the leaves in T_R will be rules in B_0. We can substitute the node that applies R with a subtree constructed from the leaves of T_R. Therefore we obtain a proof for $\Rightarrow Q$ using one less rule in $B_1 \setminus B_0$, and we can repeat this process until we obtain a proof tree with only rules in B_0.

($a \implies b$) If $\Rightarrow Q$ is obtainable by the original rules in B_0, then it is also obtainable by the saturated rules in B_1.

Let T_0 be the derivation tree for backward chaining on B_0 rules. We can construct a derivation tree T_1 using only rules in B_1 as follows: keep a frontier of rules that need to be considered, which is initially the set of leaves (closed facts). All the closed facts have empty premises, so their rules are also in B_1 and therefore can be added to T_1.

3.3 The Applied Pi-Calculus

The Applied Pi-calculus [AF01] is a process calculus stemming from the Pi-calculus [MPW92]. As in the Pi-calculus, it features the ability to create new names, and to use such names both as data and as communication channels. The distinguishing feature of the Applied Pi-calculus is its ability to encode cryptographic primitives by introducing logic terms in a fashion similar to those that we have seen in Section 3.2.1. In the following section, we will present the calculus as described in [Bla09], without events.

The syntax for the Applied Pi-calculus is shown in Figure 3.3. As in the Herbrand universe, we have terms M,N. We extend them in the process algebra to be either variables, names and constructors (functions). In contrast to Figure 3.2, names in the Applied Pi-calculus have no arguments.

$M,N ::=$	x,y,z	variables
	$\mid a,b,c$	names
	$\mid f(M_1,\ldots,M_n)$	constructors
$P,Q ::=$	0	terminated process
	$\mid !P$	replication
	$\mid P \mid Q$	parallel composition
	$\mid$ **new** a; P	restriction
	$\mid$ **in**(M,x); P	input
	$\mid$ **out**(M,N); P	output
	$\mid$ **let** $x = g(M_1,\ldots,M_n)$ **in** P **else** Q	destructor application
	$\mid$ **if** $M = N$ **then** P **else** Q	conditional

Figure 3.3: Applied Pi-calculus

In Section 3.2.1 we used equations on terms to symbolically represent cryptographic primitives. In the Applied Pi-calculus we use rewrite rules, which can be seen as a directed form of such equations. These have the form:

$$\forall \vec{x} \,.\, g(M_1,\ldots,M_n) \rightarrow M$$

where we require that $fv(M) \subseteq fv(M_1,\ldots,M_n) \subseteq \vec{x}$. We will often omit the universal quantifier in this presentation, and assume $\vec{x} = fv(M_1,\ldots,M_n)$. For example, asymmetric decryption can be represented with the following rule:

$$adec(sk(x_k), aenc(pk(x_k), x_m)) \rightarrow x_m$$

Given a rewrite rule $r = g(M_1,\ldots,M_n) \rightarrow M$, and a term $t = g(N_1,\ldots,N_n)$, there exists a rewrite relation $t \rightarrow_r t'$ iff there is a substitution $\sigma = mgu(g(M_1,\ldots,M_n), g(N_1,\ldots,N_n))$ unifying the left hand side of the rule with the term being rewritten, and $t' = \sigma(M)$. We will omit the subscript r and assume a fixed set of rewrite rules R in the remainder of this chapter.

Next we introduce the syntax for processes. Processes P,Q are: the empty process 0, the infinite replication of a process $!P$, the parallel composition $P \mid Q$ which runs two processes in parallel, the restriction **new** a; P which binds a in P to a fresh name. The primitives for communication are input **in**(M,x); P, which receives a value on channel M that will be bound to x in P, and output **out**(M,N); P, which sends N on channel M. The destructor operator **let** $x = g(M_1,\ldots,M_n)$ **in** P **else** Q applies non-deterministically a rewrite rule $r \in R$ producing the relation $g(M_1,\ldots,M_n) \rightarrow_r M$ and then executes $P\{^M/_x\}$, otherwise executes Q. Finally **if** $M = N$ **then** P **else** Q checks equality between terms M

and N and executes P if the two terms are equal, Q otherwise, and event, which signals an event in the execution of the process, marked with the term M. Note that this can be seen as syntactic sugar over **let** $x = eq(M,N)$ **in** P **else** Q, where the reduction of eq is defined by the rewrite rule $eq(x,x) \to x$.

Syntactic sugar Throughout this thesis we will use shorthand notation for standard constructs in the Applied Pi-calculus: we represent n-tuples with angle brackets notation $\langle M_1,\ldots,M_n\rangle$ and assume the presence of the projections $\pi_{i,n}(\langle x_1,\ldots,x_n\rangle) \to x_i$ for $1 \leq i \leq n$. We will use pattern matching syntax on tuple inputs and omit empty else branches, hence we write for example $\mathbf{in}(ch,x)$; $\mathbf{in}(ch,\langle =f(x),y\rangle)$; P instead of $\mathbf{in}(ch,x)$; $\mathbf{in}(ch,z)$; **let** $x' = \pi_{1,2}(z)$ **in** (**if** $x' = f(x)$ **then** (**let** $y = \pi_{2,2}(z)$ **in** P **else** 0) **else** 0) **else** 0.

3.3.1 Semantics

Next we present the semantic rules for the Applied Pi-calculus, shown in Figure 3.4. We will later link this semantics to a translation of Applied Pi-calculus processes into Horn clauses, of which we can prove the soundness of the analysis. A configuration $\mathscr{P}$ represents a multiset of concurrent processes P. Each rule $\mathscr{P} \to \mathscr{P}'$ is intended as a transition between the parallel composition of all processes in $\mathscr{P}$ to the parallel composition of all processes in $\mathscr{P}'$. That is:

$$|_{P\in\mathscr{P}} P$$

Rule NIL eliminates the empty process 0. Rule PAR splits the parallel composition of P and Q in two separate processes. Replication REPL adds a copy of P to $\mathscr{P}$, while keeping the process under replication $!P$. This allows to have infinitely many copies of the same process. Rule COM puts two processes in communication, matching the input $\mathbf{in}(M,x)$ with the corresponding output $\mathbf{out}(M,N)$. The resulting processes are $P_1\{^N/_x\}$, where the substitution replaces the input variable x with the matching output term N, and P_2. Rule NEW creates a fresh value a' that is not present in the current configuration. Rule LET-1 is applied when the rewrite rule $g(M_1,\ldots,M_n) \to M$ succeeds, and replaces x with M in P; when the rewrite rule fails, LET-2 is applied and Q the executed process. Finally, rule IF-1 transitions to P if $M = N$, and rule IF-2 transitions to Q otherwise.

In order to formalise our translation from Pi-calculus processes into Horn clauses, we switch to the term algebra of Figure 3.2 where names are constructor symbols, and we extend the syntax of replication and restriction and instrument the semantics: the new syntax for replication $!^i P$ introduces the unique replication index i; the new syntax for restriction $\mathbf{new}^l\ a;$ introduces the unique label l for the name a.

$$\mathscr{P} \uplus \{0\} \to \mathscr{P} \qquad \text{NIL}$$
$$\mathscr{P} \uplus \{P \mid Q\} \to \mathscr{P} \uplus \{P, Q\} \qquad \text{PAR}$$
$$\mathscr{P} \uplus \{!\,P\} \to \mathscr{P} \uplus \{P, !P\} \qquad \text{REP}$$
$$\mathscr{P} \uplus \{\mathbf{in}(M,x); P_1, \mathbf{out}(M,N); P_2\} \to \mathscr{P} \uplus \{P_1\{^N/_x\}, P_2\} \qquad \text{COM}$$
$$\mathscr{P} \uplus \{\mathbf{new}\ a; P\} \to \mathscr{P} \uplus \{P\{^{a'}/_a\}\} \quad \text{with } a' \text{ fresh} \qquad \text{NEW}$$
$$\mathscr{P} \uplus \{\mathbf{let}\ x = g(M_1,\ldots,M_n)\ \mathbf{in}\ P\ \mathbf{else}\ Q\} \to \mathscr{P} \uplus \{P\{^M/_x\}\} \qquad \text{LET-1}$$
$$\text{if } g(M_1,\ldots,M_n) \to M$$
$$\mathscr{P} \uplus \{\mathbf{let}\ x = g(M_1,\ldots,M_n)\ \mathbf{in}\ P\ \mathbf{else}\ Q\} \to \mathscr{P} \uplus \{Q\} \qquad \text{LET-2}$$
$$\text{if } g(M_1,\ldots,M_n) \not\to$$
$$\mathscr{P} \uplus \{\mathbf{if}\ M = N\ \mathbf{then}\ P\ \mathbf{else}\ Q\} \to \mathscr{P} \uplus \{P\} \quad \text{if } M = N \qquad \text{IF-1}$$
$$\mathscr{P} \uplus \{\mathbf{if}\ M = N\ \mathbf{then}\ P\ \mathbf{else}\ Q\} \to \mathscr{P} \uplus \{Q\} \quad \text{if } M \neq N \qquad \text{IF-2}$$

Figure 3.4: Semantics for the Applied Pi-calculus

The semantics is extended by defining $\mathscr{P}$ as the multiset of tuples of a process P and a lists of terms V, representing its environment. All the rules are trivially extended by converting $\mathscr{P} \uplus \{P\} \to \mathscr{P} \uplus \{P'\}$ into $\mathscr{P} \uplus \{(P,V)\} \to \mathscr{P} \uplus \{(P',V)\}$, except for the interesting cases of replication, restriction and communication:

$$\mathscr{P} \uplus \{(!^k P, V)\} \to \mathscr{P} \uplus \{(P, k :: V), (!^{k+1} P, V)\} \qquad \text{REP'}$$
$$\mathscr{P} \uplus \{(\mathbf{new}^l\ a; P, V)\} \to \mathscr{P} \uplus \{(P\{^{a^l[V]}/_a\})\} \qquad \text{NEW'}$$
$$\mathscr{P} \uplus \{(\mathbf{in}(M,x); P_1, V_1), (\mathbf{out}(M,N); P_2, V_2)\}$$
$$\to \mathscr{P} \uplus \{(P_1\{^N/_x\}, N :: V_1), (P_2, V_2)\} \qquad \text{COM'}$$

The rule REP' introduces an instance of P with replication index k – recorded in the environment – and increases the replication index to $k+1$ to distinguish future instances of the rule. The rule NEW'

3.3.2 Translation

As we have seen in Section 3.2.1, we can represent a security protocol specification with a set of Horn clauses. In our introduction we could only give an intuitive understanding

$$[\![0]\!]HV = \emptyset$$
$$[\![!^i P]\!]HV = [\![P]\!]H(V \cup \{x_i\})$$
$$[\![\mathbf{new}^l\ a; P]\!]HV = [\![P\{^{a^l[V]}/_a\}]\!]HV$$
$$[\![P \mid Q]\!]HV = [\![P]\!]HV \cup [\![Q]\!]HV$$
$$[\![\mathbf{in}(M,x); P]\!]HV = [\![P]\!](H \wedge \mathsf{msg}(M,x))(V \cup \{x\})$$
$$[\![\mathbf{out}(M,N); P]\!]HV = [\![P]\!]HV \cup \{H \Rightarrow \mathsf{msg}(M,N)\}$$
$$[\![\mathbf{let}\ x = g(M_1,\ldots,M_n)\ \mathbf{in}\ P\ \mathbf{else}\ Q]\!]HV =$$
$$\bigcup\{[\![P_\sigma]\!]H_\sigma V_\sigma \mid \forall \vec{x}\,.\, g(M'_1,\ldots,M'_n) \to M' \text{ is a defined rewrite rule,}$$
$$\sigma \text{ is a m.g.u. that satisfies } \bigwedge_i M_i \doteq M'_i \wedge x \doteq M'\} \cup [\![Q]\!]HV$$
$$[\![\mathbf{if}\ M = N\ \mathbf{then}\ P\ \mathbf{else}\ Q]\!]HV = \bigcup\{[\![P_\sigma]\!]H_\sigma V_\sigma \mid \sigma \in mgu(M,N)\} \cup [\![Q]\!]HV$$

Figure 3.5: Translation of Applied Pi-calculus processes into Horn clauses

of how to construct this logical specification. Now we have a formal specification language — the Applied Pi-calculus — and a formal semantics, hence we can construct an automatic translation from a protocol specification into a set of Horn clauses. As the Applied Pi-calculus uses terms as channels, we extend our Horn clause representation of Section 3.2.1 with a new predicate, $\mathsf{msg}(M,N)$, used to record that message N is present in channel M. This allows to encode secret channels, as we will see by the rules.

Figure 3.5 presents the rules for translating each construct of the calculus into a set of Horn clauses. The translation function $[\![P]\!]HV$ takes as parameters the process P to be translated, an increasing set of hypotheses H, and a set of terms V that are either session identifiers (variables), or accumulated inputs to the current process.

The translation for the stuck process 0 produces an empty set of clauses. The translation for replication $!^i P$ introduces the session identifier i as a variable x_i in V, and translates the continuation with the new parameters. Restriction $\mathbf{new}^l\ a; P$ introduces a substitution for P, where a is replaced with a name $a^l[V]$ that depends on the terms in V and the label l. Parallel composition $P \mid Q$ produces the union of the clauses for the translation of the two processes P and Q. The rule for input $\mathbf{in}(M,x); P$ adds a hypothesis of the form $\mathsf{msg}(M,x)$, hence requiring the presence of a message x on channel M to proceed with P; it also adds x to the set of influencing terms. The rule for output $\mathbf{out}(M,N); P$ produces a clause with H as hypotheses, and the predicate $\mathsf{msg}(M,N)$ as conclusion. Intuitively this indicates that if all inputs (hypotheses in H) are satisfied, then the message N will be sent on channel M. Finally the rule for $\mathbf{let}\ x = g(M_1,\ldots,M_n)\ \mathbf{in}\ P\ \mathbf{else}\ Q$ takes an applicable rewrite rule $g(M'_1,\ldots,M'_n) \to M'$ and computes the most general unifier

σ that satisfies the equality constraints $M_i \stackrel{?}{=} M'_i$ and $x \stackrel{?}{=} M'$. This produces a most general substitution σ that is applied to the resulting process P. Similarly the rule for **if** computes the unification σ between M and N, if one exists, and applies it as a substitution for the continuation P. As an over-approximation, both in the rule for **let** and the rule for **if**, the process Q in the **else** branch has no applied constraint.

For example, the process **if** $a = a$ **then** 0 **else out**$(c, secret)$; 0, assuming the environment that restricts all the free variables, gets translated to the clause: $\Rightarrow \mathsf{msg}(c[], secret[])$. Hence the fact $\mathsf{msg}(c[], secret[])$ is reachable from the generated Horn clauses, but *secret* can never be sent through channel c, according to the semantic rules. Here lies the strength, as well as the weakness, of this translation: some facts will be reachable that do not correspond to the system's behaviour, however all this behaviour is captured by a reachable fact (as we will show soon in Section 3.3.3) and we avoid a potential source of non-termination [].

Standard attacker predicates Next we define a fixed set of rules $\mathscr{C}_A$, representing the capabilities of the attacker.

The attacker can send any message he knows:

$$\mathsf{att}(x_c) \wedge \mathsf{att}(x_m) \Rightarrow \mathsf{msg}(x_c, x_m)$$

The attacker can read on known channels:

$$\mathsf{msg}(x_c, x_m) \wedge \mathsf{att}(x_c) \Rightarrow \mathsf{att}(x_m)$$

The attacker can apply any knwon n-ary constructor f on known data:

$$\mathsf{att}(x_1) \wedge \cdots \wedge \mathsf{att}(x_n) \Rightarrow \mathsf{att}(f(x_1, \ldots, x_n))$$

The attacker can apply any public destructor $\forall \vec{x} \,.\, g(M_1, \ldots, M_n) \rightarrow M$:

$$\mathsf{att}(M_1) \wedge \cdots \wedge \mathsf{att}(M_n) \Rightarrow \mathsf{att}(M)$$

Finally, the attacker knows the public channel shared with the protocol and at least a name, to which we assign a special label A for the attacker:

$$\Rightarrow \mathsf{att}(ch[]) \qquad\qquad \Rightarrow \mathsf{att}(a^A[x_i])$$

3.3.3 Soundness

The translation that we presented in the previous section allows to analyse a process P, in the presence of an attacker A, who interacts by means of the public channel ch. This result is achieved by means of the following theorem.

THEOREM 2 (SOUNDNESS OF THE TRANSLATION) *Let P and A be two processes, representing the protocol and the attacker – where* **new** *ch*; $(P|A)$ *is closed, let* $Q = \mathsf{msg}(M,N)$ *be a reachability query, let* $\mathscr{C}_P = [\![P]\!]\emptyset[ch \mapsto ch[]]$ *be the translation of the protocol P into Horn clauses, and let $\mathscr{C}_A$ be the fixed set of rules for the attacker.*

If $\{\textbf{new } ch; (P|A)\} \rightarrow^* \mathscr{P} \uplus \{\textbf{out}(M',N'); P'\}$ *and there exists* σ *m.g.u. satisfying* $M \stackrel{\circ}{=} M'$ *and* $N \stackrel{\circ}{=} N'$*, then the set of facts* $\mathscr{F}_P$ *derivable from the saturation* $sat(\mathscr{C}_P \cup \mathscr{C}_A)$ *contains* $\mathsf{msg}(M,N)_\sigma$*. In particular, if* $\mathsf{att}(M)_\sigma$*, then also* $\mathsf{att}(N)_\sigma$*.*

The proof of this theorem relies on the construction of a typing judgement for processes, which we are going to introduce for our extension of the Applied Pi-calculus in Section 6.7. Essentially, this typing judgement establishes a relation between the semantics and the set of clauses $\mathscr{C}_P \cup \mathscr{C}_A$, ensuring that whenever an output $\textbf{out}(M,N)$ is derivable from the initial configuration, the corresponding predicate $\mathsf{msg}(M,N)$ is also reachable in the set of saturated facts $\mathscr{F}_P$. This relation is established both between the protocol P and the set of clauses $\mathscr{C}_P$, and between any arbitrary attacker process A with access to the public channel and the fixed set of clauses $\mathscr{C}_A$.

As a direct consequence of Theorem 2 we have the following fact: if $\mathsf{msg}(M,N) \notin \mathscr{F}_P$, then for any substitution σ with $Dom(\sigma) \subseteq fv(M,N)$ the output $\textbf{out}(M_\sigma, N_\sigma,)$ is unreachable from the initial configuration.

3.4 Computational Model

The symbolic representation of security protocols that we presented in Section 3.2 makes simplifying assumptions on reality. For example, the rewrite rule $sdec(k, senc(k,m)) \rightarrow m$ does not represent all the possible behaviours of an encryption function. This representation excludes for example that the attacker can change specific bits of the ciphertext knowing the structure of the messages being sent, and does meaningful manipulations on encrypted data.

Consider the case where A sends to B an encrypted transaction:

$$A \rightarrow B : senc(k_{AB}, \langle A, B, \mathsf{send}, 100\rangle)$$

If the intruder knows the structure of the message, she can try to insert another value at the end to gain an advantage, but only if the encryption is *malleable*: that is, only if the intruder can transform a valid ciphertext into another valid ciphertext.

In the computational model, the attacker is a deterministic Turing machine operating on bitstrings, and cryptographic functions are computable in polynomial time, while

their inverses are assumed to be hard to compute (e.g., not deterministic polynomial time functions). A security proof in this model consists of a *sequence of games* [Sho04, BR04] — probabilistic programs where the attacker is allowed to freely interact with the protocol — where the initial game is the real protocol, and each successive game is the result of a transformation of the previous probabilistic program, where the distance in the probability distribution and the traces generated by the two games is *negligible*. Here negligible means bounded by a number smaller than the inverse of any polynomial in the security parameter — e.g., the size of a key, nonce etc.

The original game (program) must contain a *failure event* — for example an event that occurs when the attacker can forge a MAC signature, or distinguish between two different ciphertexts. Each transformation has associated a difference in probability of reaching such failure event, and the final game is one such that the probability of failure is very easily computed, most often being 0, 1 or $\frac{1}{2}$. If the sequence is represented as:

$$\mathscr{G}_0 \xrightarrow{p_1} \mathscr{G}_1 \xrightarrow{p_2} \mathscr{G}_2 \ldots \xrightarrow{p_n} \mathscr{G}_n$$

then the proof gives a probability bound to attacking the system that is the sum of all the distances:

$$\sum_{i=1}^{n} p_i$$

3.4.1 Building Blocks

Here we mention some of the most common security properties that are used in computational proofs. The first three properties (EAV, IND-CPA, IND-CCA) present different security requirements for encryption, and the last (INT-CMA) defines the requirements for secure message authentication codes.

Eavesdropping Security (EAV)

1. The attacker sends two messages m_0 and m_1 to the protocol
2. The protocol samples a random bit $b \xleftarrow{\$} \{0,1\}$, then encodes $c \leftarrow senc(k, m_b)$ and gives it to the attacker.
3. The attacker guesses which plaintext $m_{b'}$ has been encrypted, and outputs b'.
4. If $b = b'$ then the attacker won the game.

This game formalises the ability of the attacker to gain information from one encryption, and relate it to one of the two original plaintexts m_0, m_1. If the attacker can win the game with probability that is non-negligibly higher than $\frac{1}{2}$, then it has an advantage over a purely random guess. Hence the attacker can recover some information about the plaintext. If the probability that the attacker wins the game is bounded by $\frac{1}{2}$ plus the inverse of any polynomial $p(n)$ on the security parameter n, then the scheme is defined to be IND-EAV secure.

Note that the attacker can choose both messages, including their length. The scheme should hide this information. However an encryption that is secure by this definition can still reveal information about the message.

If the encryption is deterministic, then an attacker is able to check whether two ciphertexts $c_0 = senc(k, m_0)$ and $c_1 = senc(k, m_1)$ are the encryption of the same plaintext, simply by comparing them: $c_0 = c_1 \implies senc(k, m_0) = senc(k, m_1)$, because the encryption is a bijective function.

Indistinguishability under Chosen Plaintext Attack (IND-CPA)

1. The protocol generates a random key k.
2. The attacker has access to the encryption oracle $senc(k, \cdot)$, and outputs a pair of messages m_0, m_1 of the same length.
3. The protocol chooses a random bit $b \xleftarrow{\$} \{0,1\}$ and then computes the encryption $c \leftarrow senc(k, m_b)$ and gives it to the attacker.
4. The attacker continues to have access to the oracle $senc(k, \cdot)$, and guesses which plaintext has been encrypted by outputting b'.
5. If $b = b'$ then the attacker won the game.

Similarly to the definition of eavesdropping security, the encryption scheme is IND-CPA secure if the attacker is not able to win the game with probability non-negligibly higher than $\frac{1}{2}$. That is, the attacker does not have a better strategy than a purely random guess.

This definition differs from eavesdropping security in that the attacker has access to the encryption oracle, before and after receiving the encrypted text c. An encryption scheme that satisfies this definition must behave randomly. Encrypting two times the same plaintext m with the same key k results in two different ciphertexts, otherwise the attacker could ask the oracle to encrypt one of the two plaintexts (e.g. m_0) and compare it to the challenge.

Indistinguishability under Chosen Ciphertext Attacks (IND-CCA)

1. The protocol generates a random key k.
2. The attacker has access to the encryption oracle $senc(k,\cdot)$ *and* the decryption oracle $sdec(k,\cdot)$, and outputs a pair of messages m_0, m_1 of the same length.
3. The protocol chooses a random bit $b \xleftarrow{\$} \{0,1\}$, encrypts m_b as $c \leftarrow senc(k,m_b)$, and outputs the result.
4. The attacker continues to have access to the encryption and the decryption oracles $senc(k,\cdot)$ and $sdec(k,\cdot)$, but is not allowed to decrypt the ciphertext c received by the protocol. However, it cannot use the decryption oracle on the challenge ciphertext c. With this information, the attacker has to guess which message has been encrypted, and outputs a bit b'.
5. If $b = b'$ then the attacker has won the game.

Again, similarly to the previous definitions, an encryption scheme is IND-CCA secure if the probability that the attacker wins the game is bounded by $\frac{1}{2}$ plus the inverse of any polynomial $p(n)$ on the security parameter n. That is, the attacker has no better strategy than making a purely random guess.

This definition strengthens IND-CPA security by giving the attacker access to the decryption oracle $sdec(k,\cdot)$, with the sole requirement that the challenge ciphertext c cannot be decrypted. Note that an encryption scheme that allows *meaningful transformations* on encrypted data would not be IND-CCA secure. If the attacker can modify the challenge c into a valid ciphertext c' — flipping one bit, for example — then c' can be decoded into m' by the decryption oracle, since it is not the original challenge. This would give the attacker a non-negligible advantage, by learning a plaintext m' that is meaningfully related to one of the two initial plaintexts m_1 and m_2. Therefore, IND-CCA resistant schemes are also said to be *non-malleable*.

Integrity under Chosen Message Attacks (INT-CMA)

1. The protocol generates a random key k.
2. The attacker has access to the MAC oracle $mac(k,\cdot)$ and outputs a pair (m,t) of a message and a tag. Let $\mathcal{Q}$ be the set of queries asked by the attacker to the MAC oracle.
3. If $verify(k,m,t) = 1$, hence verification succeeds, and $m \notin \mathcal{Q}$, then the attacker wins the game.

A MAC code is INT-CMA secure if the probability that the attacker wins the game is negligible.

Tool Support As discussed, the game-playing technique can be formalised as a set of program transformations. Halevi advocated about a decade ago the need for computer supported security proofs [Hal05]. The formal methods community took up the challenge, and has recently come up with formal approaches that support the cryptographer in their security proofs. To construct such a tool one needs a formal semantics in which these games can be expressed, and a technique to reason about the games.

Barthe et al. propose the EasyCrypt framework [BGHB11, BGZB09], which uses a probabilistic extension of the WHILE language — called pWHILE — and the probabilistic relational Hoare logic to reason about program transformation. The probabilistic relational Hoare logic is itself an extension of Hoare logic with judgements of the form $\{P\}G_1 \sim_p G_2\{Q\}$. In one such judgement G_1 and G_2 are the programs (games) and P is a precondition on the programs that must hold before the execution G_1 and G_2. The programs G_1 and G_2 satisfy the judgement if they both terminate and satisfy Q with a probability that is bounded by p. This type of judgement closely maps the structure that we have seen in this section.

Another emerging approach is the one employed by CryptoVerif [Bla07], with a language similar to the Applied Pi-calculus, which employs a probabilistic semantics and a fixed set of game transformations to automatically derive a proof of security from the original game representing the protocol.

Finally we mention the F* project [SCF+13], which we are going to use in Chapter 7 of this thesis, and its previous incarnations FINE and F7 [CCS10, BBF+11]. F* is an extension of the ML family of functional languages with refinement types, types with attached formulas on data, that can be used to construct security proofs around protocol code.

3.5 Conclusion

In this chapter we have seen two different models for verifying security protocols: the symbolic Dolev-Yao model and the computational model. Both models are amenable to the application of semi- or fully-automated formal verification techniques, and their features imply trade-offs over the expressive power vs. the complexity of the verification approach.

CHAPTER 4

Formal Analysis of Authenticating CAN Protocols

As we saw in Chapter 2, the CAN bus is an embedded real-time network protocol that cannot rely on off-the-shelf schemes for authentication, because of the bandwidth limitations imposed by the network. As a result, both academia and industry proposed custom protocols that meet the particular constraints of this network, with solutions that may be deemed insecure if considered out of context.

With this chapter we set to analyse two proposed security protocols for CAN — MaCAN and CANAuth — using ProVerif. Although there are plenty of different proposals being developed in the last five years, for our analysis we selected the only two that, by their requirements, were plausible candidates for adoption by our industrial partners in SESAMO[1].

[1] `http://sesamo-project.eu`

4.1 Introduction

In this chapter we analyse the MaCAN and CANAuth protocols described in [HRS12] and [HSV11], with the assistance of the ProVerif protocol verifier [BS13].

Our analysis of MaCAN shows two flaws in the specification, one in the key distribution scheme, and another in signal authentication.

The first flaw allows the initiating principal to believe that a session has been established, while the other parties have not received a session key. In MaCAN, key distribution happens between three or more parties: an *initiator*, responsible for starting the procedure, the *key server*, responsible for delivering the session key, and one or more *responders*, which also need to obtain the session key from the key server.

The slightly asymmetric behaviour of the protocol allows an attacker to reuse the signature of the acknowledgement message sent by the initiator, in order to simulate an acknowledgement message coming from the responder, therefore completing authentication on one side. Furthermore, the attacker can manipulate the behaviour of the key server so that the responder never receives a request for authentication, and therefore the responder is never activated. This leads to an incomplete session establishment where one of the parties believes that it can communicate authenticated messages while the other will refuse such messages because it is not in possession of a valid session key.

Our proposed correction removes the asymmetry in the two phases of the protocol, and prevents the attack. We model our modification of the MaCAN protocol in ProVerif and discover another minor problem in the format of the acknowledgement message, that allows the attacker to successfully send acknowledgements with the signature of another principal in group sessions. Adding source information to the signature overcomes this problem, and allows us to prove the desired authentication property in the key establishment phase.

The second flaw allows repurposing an authenticated signal when a specific message format is used. An attacker can forge the signal number without this being detected, allowing, for example, the message with meaning “speed is 25” to be modified to “temperature is 25”.

Our correction modifies the signature so that the signal number is considered, preventing that particular attack form happening. However, the nature of the protocol allows replays within the validity time frame of a message, which can be rather long for its applications. Here we contribute with a discussion that clarifies which properties an application designer can expect from MaCAN, and needs to take into consideration when designing a system.

Next we analyse the CANAuth protocol with ProVerif. While the analysis reveals a potential attack in the authentication, we argue that it is a false positive, due to the limitations of the analysis provided by ProVerif, which overapproximates by abstracting away state information. These considerations pave the way for the work presented in Chapter 6, where we extend the applied pi-calculus into a language — that we call Set-Pi — with explicit stateful constructs. Set-Pi allows in this case a more faithful representation of the protocol, and a more precise analysis that proves the desired authentication property in our models. Set-Pi models of the two CAN protocols that do not suffer from such over-approximations are shown in Appendix B.3.

4.2 Formal Analysis of MaCAN

4.2.1 Key Establishment

Figure 4.1 shows our model of the MaCAN authentication procedure in the Applied Pi-calculus. All communication happens on a broadcast channel c, while we use a private channel psk for the key server to store the long term keys of the ECUs. The process KS represents the key server, ECU_i is the initiator process and ECU_j is the responder process.

Due to the abstractions introduced by ProVerif, we have to change some important aspects of the protocol in order to obtain a precise analysis. First and foremost, we remove timestamps from signatures, because ProVerif abstracts away the concept of state in its translation of processes to Horn clauses. Then we treat $group\_field$ as a name instead of a bit vector in order to simplify the models. Finally we encode long encrypted messages that would be split into multiple frame as a single message. We take these changes into consideration when we interpret the results of our analysis and we argue to which extent they introduce overapproximations.

Current MaCAN configurations have clock rates of 1 second, so it is safe to assume that timestamps can be treated as constants, since the key establishment procedure can complete within a single clock tick. Note that it is undesirable to have high clock rates due to the following constraint: the receiving end of an authenticated signal needs to check a signature against all valid timestamps within the possible reception window of the message. Therefore, increasing the clock rate also requires more computation on the receiving end, which in turn increases the worst case response time for a signal transmission. The length of the reception window for a message can be obtained with schedulability analysis [DBBL07] and depends on the number of higher priority messages that can delay the transmission of the message in question. In Chapter 5 we adapt the schedulability analysis results for the CAN bus [DBBL07] to include the costs

$KS \triangleq \mathbf{in}(c, \langle i, =\mathsf{CH}, =ks, c_i, j\rangle);$
$\mathbf{in}(psk, \langle =i, k_i\rangle);$
$\mathbf{new}\ sk_{ij};$
$\mathbf{event}\ sessk_i(i, j, c_i, sk_{ij});$
$\mathbf{out}(c, \langle ks, \mathsf{SK}, i, senc(k_i, \langle c_i, j, i, sk_{ij}\rangle)\rangle);$
$\mathbf{out}(c, \langle ks, \mathsf{RC}, j\rangle);$
$\mathbf{in}(c, \langle =j, =\mathsf{CH}, =ks, c_j, =i\rangle);$
$\mathbf{in}(psk, \langle =j, k_j\rangle);$
$\mathbf{event}\ sessk_j(j, i, c_i, sk_{ij});$
$\mathbf{out}(c, \langle ks, \mathsf{SK}, j, senc(k_j, \langle c_j, i, j, sk_{ij}\rangle)\rangle);$
$\mathbf{in}(c, \langle =sk_{ij}\rangle);$
$\mathbf{event}\ revealed(sk_{ij}); 0$

$ECU_i \triangleq \mathbf{new}\ c_i;$
$\mathbf{event}\ authStart_i(i, j, c_i);$
$\mathbf{out}(c, \langle i, \mathsf{CH}, ks, c_i, j\rangle);$
$\mathbf{in}(c, \langle =ks, =\mathsf{SK}, =i, resp\rangle);$
$\mathbf{let}\ \langle =c_i, =j, =i, sk_{ij}\rangle = sdec(k_i, resp)\ \mathbf{in}$
$\mathbf{event}\ authAck_i(i, j, c_i, sk_{ij});$
$\mathbf{out}(c, \langle i, \mathsf{ACK}, j, cmac(sk_{ij}, j, AK)\rangle);$
$\mathbf{in}(c, \langle =j, =\mathsf{ACK}, =i, =cmac(sk_{ij}, j, \mathsf{ACK})\rangle);$
$\mathbf{event}\ authEnd_i(i, j, c_i, sk_{ij}); 0$

$ECU_j \triangleq \mathbf{in}(c, \langle =ks, =\mathsf{RC}, =j\rangle);$
$\mathbf{in}(c, \langle i, =\mathsf{ACK}, =j, ack\rangle);\ \mathbf{new}\ c_j;$
$\mathbf{event}\ authStart_j(j, i, c_j);$
$\mathbf{out}(c, \langle j, \mathsf{CH}, ks, c_j, i\rangle);$
$\mathbf{in}(c, \langle =ks, =\mathsf{SK}, =j, resp\rangle);$
$\mathbf{let}\ \langle =c_j, =i, =j, sk_{ij}\rangle = sdec(k_j, resp)\ \mathbf{in}$
$\mathbf{if}\ ack = cmac(sk_{ij}, j, \mathsf{ACK})\ \mathbf{then}$
$\mathbf{event}\ authAck_j(j, i, c_j, sk_{ij});$
$\mathbf{out}(c, \langle j, AK, i, cmac(sk_{ij}, j, \mathsf{ACK})\rangle);$
$\mathbf{event}\ authEnd_j(j, i, c_j, sk_{ij}); 0$

Figure 4.1: MaCAN key establishment process in the Applied Pi-calculus

of preparing and then checking signatures.

In Figure 4.1, *KS* represent the key server process. It waits on the public channel for a challenge c_i to establish a session between ECU_i and ECU_j, retrieves k_i from its database, produces a fresh session key sk_{ij}, outputs the encoding of the session and sends a request for challenge to ECU_j. It then waits for a challenge c_j from ECU_j, retrieves its key k_j, and encodes the session key sk_{ij} in a message for ECU_j that includes the challenge c_j. Finally, it waits for the session key to be sent in clear text on the channel to signal the event $revealed(sk_{ik})$. If the event is not reachable then the secrecy of sk_{ij} is guaranteed.

ECU_i creates a new challenge c_i, sends the challenge to the key server, waits for the response of the key server and decodes the message to retrieve the session key sk_{ij}. ECU_i then sends an acknowledgement to ECU_j signed with sk_{ij}, and waits for a similar acknowledgement from ECU_j to conclude the key establishment procedure.

ECU_j waits for a request for challenge from the key server, reads the acknowledgement from ECU_i, sends its challenge to the key server, receives the session key sk_{ij}, verifies the validity of the acknowledgement from the other party and finally sends its own acknowledgement, concluding its part of the procedure.

Analysis results We analysed the following five properties for key establishment:

(i) the secrecy of long term keys k_i, k_j,

(ii) the secrecy of session keys ks_{ij},

(iii) the agreement between the events $authStart_i(i,j,c_i)$, $sessk_i(i,j,c_i,sk_{ij})$, $authAck_i(i,j,c_i,sk_{ij})$, $authEnd_i(i,j,c_i,sk_{ij})$, and

(iv) the agreement between the events $authStart_j(j,i,c_j)$, $sessk_j(j,i,c_j,sk_{ij})$, $authAck_j(j,i,c_j,sk_{ij})$, $authEnd_j(j,i,c_j,sk_{ij})$.

Using ProVerif, we were able to verify the secrecy properties *(i,ii)*, but we found a counterexample for the event correspondence *(iii)*, where an attacker can run the protocol in such a way that ECU_i receives the proper session key from message (2.12) instead of (2.8), leaving the ECU_j unauthenticated. The correspondence *(iv)* for ECU_j is proven, therefore it can only authenticate as intended by the protocol.

Figure 4.2 shows our reconstruction of the attack trace produced by ProVerif for the query of events related to ECU_i (property *iii*), thereby providing feedback to the protocol designer about how to amend the protocol. In this trace "$\rightsquigarrow$" represents a message

$$\textbf{event } authStart_i(id_i, id_j, C_i); \tag{4.1}$$
$$ECU_i \rightsquigarrow KS : \mathsf{CH}, C_i, id_j \tag{4.2}$$
$$\mathrm{M}[ECU_j] \rightarrow KS : \mathsf{CH}, a, id_i \tag{4.3}$$
$$\textbf{event } sessk_i(id_j, id_i, a, SK_{i,j}); \tag{4.4}$$
$$KS \rightarrow ECU_j : \mathsf{SK}, senc(K_{ks,j}, \langle a, id_i, id_j, SK_{i,j}\rangle) \tag{4.5}$$
$$KS \rightsquigarrow ECU_i : \mathsf{RC} \tag{4.6}$$
$$\mathrm{M}[ECU_i] \rightarrow KS : \mathsf{CH}, C_i, id_j \tag{4.7}$$
$$\textbf{event } sessk_j(id_i, id_j, C_i, SK_{i,j}); \tag{4.8}$$
$$KS \rightarrow ECU_i : \mathsf{SK}, senc(K_{ks,i}, \langle C_i, id_j, id_i, SK_{i,j}\rangle) \tag{4.9}$$
$$\textbf{event } authAck_i(id_i, id_j, SK_{i,j}); \tag{4.10}$$
$$ECU_i \rightarrow ECU_j : \mathsf{ACK}, cmac(SK_{i,j}, \langle T, id_j, group\_field\rangle) \tag{4.11}$$
$$M[ECU_j] \rightarrow ECU_i : \mathsf{ACK}, cmac(SK_{i,j}, \langle T, id_j, group\_field\rangle) \tag{4.12}$$
$$\textbf{event } authEnd_i(id_i, id_j, SK_{i,j}); \tag{4.13}$$

Figure 4.2: Attack trace to MaCAN key establishment

deleted by the attacker (this can be achieved by jamming the signal at the proper time or by making one of the participating nodes or an involved CAN gateway unavailable) and $\mathrm{M}[x]$ represent the malicious agent impersonating x (it can be done by sending a message with the proper CAN-ID).

This attack relies on the possibility to remove messages from the channel. The attacker learns the current challenge and the destination ID (4.2), while suppressing the message. It then impersonates ECU_j and starts sending a random challenge (4.3), initiating the communication with the key server in the opposite direction. The key server then sends a legitimate message to ECU_j (4.5), who will ignore it as it did not request a session key. Then the key server sends a request for challenge to ECU_i (4.6), which may be suppressed by the attacker. The attacker remembers the previous challenge from ECU_i and replays it on the key server (4.7), receiving the session key encrypted for ECU_i in return (4.9). Finally ECU_i sends its acknowledgement message (4.11), and since the form of the two acknowledgement messages is the same for ECU_i (2.10) and ECU_j (2.13), the attacker can impersonate ECU_j and send back the same signature (4.12) so that ECU_i believes that also ECU_j is authenticated.

Corrected model We propose a correction of the model where the asymmetries that cause the improper authentication behaviour are removed. Figure 4.3 shows the corrected procedure.

$$ECU_i \rightarrow KS : \mathsf{CH}, C_i, id_j \tag{4.14}$$
$$KS \rightarrow ECU_i : \mathsf{SK}, senc(K_{i,ks}, \langle C_i, id_j, id_i, SK_{i,j} \rangle) \tag{4.15}$$
$$KS \rightarrow ECU_j : \mathsf{RC} \tag{4.16}$$
$$ECU_i \rightarrow ECU_j : \mathsf{ACK}, group\_field, cmac(SK_{i,j}, \langle T, id_i, id_j, group\_field \rangle) \tag{4.17}$$
$$ECU_j \rightarrow KS : \mathsf{CH}, C_j, id_i \tag{4.18}$$
$$KS \rightarrow ECU_j : \mathsf{SK}, senc(K_{j,ks}, \langle C_j, id_i, id_j, SK_{i,j} \rangle) \tag{4.19}$$
$$ECU_j \rightarrow ECU_i : \mathsf{ACK}, group\_field, cmac(SK_{i,j}, \langle T, id_j, id_i, group\_field \rangle) \tag{4.20}$$

Figure 4.3: Modified MaCAN key establishment procedure

To guarantee the agreement property we modify the form of the acknowledgement message. The CMAC signature is now using the current timestamp, the source and the destination of the message as content. Since CMAC is a hashed signature, adding more parameters to the function does not affect the final payload size, therefore the modified protocol still fits the space constraints of CAN. The two acknowledgement messages (4.17) and (4.20) are now symmetrical. We added the source information on the signed hashes, as well as the destination. This allows not only to prove the necessary correspondence for two-party sessions, but in case of group sessions it removes the chance for an intruder to reuse an acknowledgement message of another principal.

Figure 4.4 shows the corrected model in the Applied Pi-calculus, where we applied the modified behaviour for the three processes. The properties *(i–iv)* that we defined in Section 4.2.1 have all been proved in this model.

4.2.2 MaCAN message authentication

During a session, authenticated parties can send authenticated signals. As described in Section 2.3.2, the transmission of an authenticated signal needs to follow a specific request (2.15). Depending on whether the authenticated signal fits in 32 bits — that is half of the available CAN payload size — the responding ECU uses either message format (2.17) or (2.16).

Figure 4.5 shows two communicating processes that exchange authenticated messages according to message format (2.16). ECU_i requests an authenticated signal with the first output according to (2.15). Then it keeps waiting for an authenticated signal and checks whether the signature corresponds to its own computation of it, marking with an *accept* the acceptance of an authenticated signal. On the other side ECU_j receives a request for authentication, checks its signature and starts sending signals, marking with a *send*

$KS \triangleq$ **in**$(c, \langle i, =\mathsf{CH}, =ks, c_i, j\rangle)$;
in$(psk, \langle =i, k_i\rangle)$;
new sk_{ij};
event $sessk_i(i, j, c_i, sk_{ij})$;
out$(c, \langle ks, \mathsf{SK}, i, senc(k_i, \langle c_i, j, i, sk_{ij}\rangle)\rangle)$;
out$(c, \langle ks, \mathsf{RC}, j, i\rangle)$;
in$(c, \langle =j, =\mathsf{CH}, =ks, c_j, =i\rangle)$;
in$(psk, \langle =j, k_j\rangle)$;
event $sessk_j(j, i, c_i, sk_{ij})$;
out$(c, \langle ks, \mathsf{SK}, j, senc(k_j, \langle c_j, i, j, sk_{ij}\rangle)\rangle)$;
in$(c, =sk_{ij})$;
event $revealed(sk_{ij})$; 0

$ECU_i \triangleq$ **new** c_i;
event $authStart_i(i, j, c_i)$;
out$(c, \langle i, \mathsf{CH}, ks, c_i, j\rangle)$;
in$(c, \langle =ks, =\mathsf{SK}, =i, resp\rangle)$;
let $\langle =c_i, =j, =i, sk_{ij}\rangle = sdec(resp, k_i)$ **in**
event $authAck_i(i, j, c_i, sk_{ij})$;
out$(c, \langle i, \mathsf{ACK}, j, cmac(sk_{ij}, i, j, AK)\rangle)$;
in$(c, =j, =\mathsf{ACK}, =i, =cmac(sk_{ij}, j, i, \mathsf{ACK}))$;
event $authEnd_i(i, j, c_i, sk_{ij})$; 0

$ECU_j \triangleq$ **in**$(c, \langle =ks, =\mathsf{RC}, =j, i\rangle)$;
new c_j; **event** $authStart_j(j, i, c_j)$;
out$(c, \langle j, \mathsf{CH}, ks, c_j, i\rangle)$;
in$(c, \langle i, =\mathsf{ACK}, =j, ack\rangle)$;
in$(c, \langle =ks, =\mathsf{SK}, =j, resp\rangle)$;
let $\langle =c_j, =i, =j, sk_{ij}\rangle = sdec(resp, k_j)$ **in**
if $ack = cmac(sk_{ij}, i, j, \mathsf{ACK})$ **then event** $authAck_j(j, i, c_j, sk_{ij})$;
out$(c, \langle j, AK, i, cmac(sk_{ij}, j, i, \mathsf{ACK})\rangle)$; **event** $authEnd_j(j, i, c_j, sk_{ij})$; 0

Figure 4.4: Fixed version of the MaCAN key establishment process in the Applied Pi-calculus

$$
\begin{aligned}
ECU_i \triangleq & \mathbf{out}(c, \langle i, \mathsf{SIG\text{-}AUTH\text{-}REQ}, j, sig\#, n0, cmac(sk_{ij}, \langle i, j, sig\#, n0\rangle)\rangle); \\
& !(\mathbf{in}(c, \langle =j, =\mathsf{SIG\text{-}AUTH}, =i, sig\#, signal, x_{sig}\rangle); \\
& \quad \mathbf{if}\ x_{sig} = cmac(sk_{ij}, \langle i, j, signal\rangle)\ \mathbf{then} \\
& \quad \mathbf{event}\ accept(sig\#, signal); 0)
\end{aligned}
$$

$$
\begin{aligned}
ECU_j \triangleq & \mathbf{in}(c, \langle i, =\mathsf{SIG\text{-}AUTH\text{-}REQ}, =j, sig\#, prescaler, x_s\rangle); \\
& \mathbf{if}\ x_s = cmac(k_{ij}, \langle i, j, sig\#, prescaler\rangle)\ \mathbf{then} \\
& !(\mathbf{new}\ signal; \mathbf{event}\ send(sig\#, signal); \\
& \quad \mathbf{out}(c, \langle j, \mathsf{SIG\text{-}AUTH}, i, sig\#, signal, cmac(sk_{ij}, \langle i, j, signal\rangle)\rangle); 0)
\end{aligned}
$$

Figure 4.5: MaCAN message authentication processes in the Applied Pi-calculus.

event the transmission of a fresh signal.

The original paper [HRS12] is not clear about whether the CMAC signature includes the signal number. The process in Figure 4.5 does not include the signal number as part of the signature for signals. Thus, the correspondence between $send(sig\#, signal)$ and $accept(sig\#, signal)$ is not verified. An attacker can read a signed signal with a $sig\#$ value re-transmit the signal with a different $sig\#$ if multiple $sig\#$ have been requested within the same session.

A simple solution to this problem is to add $sig\#$ as part of the signature. With the modified process, which we omit for sake of brevity, we are able to verify the agreement between the events $send(sig\#, signal)$ and $accept(sig\#, signal)$.

Still we fail to verify an injective agreement between the two events. Given our specification it is possible to accept twice the same message, and this could constitute a freshness violation. Our abstraction removes all timestamps, as the modelling technology cannot efficiently deal with them, and we previously argued that they can be ignored and treated as constant within their validity window.

As current configurations have clock rates of one second, this constitutes a potentially serious flaw in the protocol. Imagine MaCAN authenticating messages for the brake control unit of a vehicle. In case of emergency braking at high speed, the driver might be tempted to go all the way down with the foot on the brake pedal, activating the ABS. The ABS control unit works by sending messages to the brake control unit, releasing the brakes at a fast interval, so that the wheels do not slide on the ground, reducing their grip. In this example an attacker could wait for a “release message” from the ABS control unit, and replay it for its whole validity, therefore effectively disabling the brake

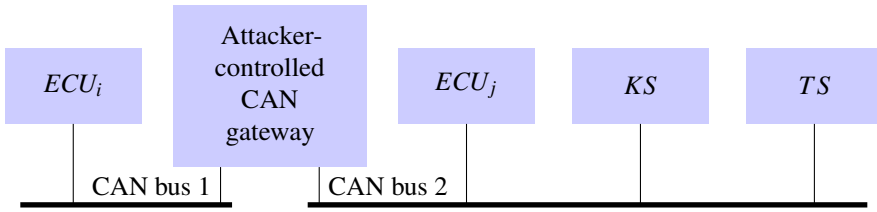


Figure 4.6: Experimental setup

for an entire second in a dangerous situation.

Given the restrictions imposed by the CAN bus, we believe that MaCAN constitutes a good enough solution for authenticating signals. A better level of security can be achieved by incrementing the clock rate. This would reduce the time window available for replaying messages, and therefore reduce the potential effect of such replay. In case of fast control loops — where a specific signal needs to be sent every 50 ms, for example — a solution that completely prevents replay attacks would synchronise the clock with the message rate, and refuse any message signed with a timestamp that has been previously used. Specific care would then be required for synchronising the clock between the communicating devices, and to avoid any unavailability issues due to improper synchronisation.

4.2.3 Experimental Evaluation

We compared the result of our analysis with the implementation of MaCAN, developed independently by our SESAMO partners at the Czech Technical University – Industrial Informatics Group, which is available under an opensource license on Github[2]. We implemented the attacks to the key establishment and message authentication procedure, putting the attacker in control of a gateway as shown in Figure 4.6. This was demonstrated to be possible in practice by Checkoway et al. [CMK+11], and representative of typical automotive architecture that we presented in Chapter 2.

The attack on key establishment was possible only after aligning the implementation to the specification contained in the paper, as the necessary acknowledgement was already corrected in our implementation. We cannot trace back, however, whether this correction was due to explicit considerations by our skilled engineers, or it happened by chance by misinterpreting the flawed specification.

Our implementation also accepted authenticated acknowledgement messages replayed

[2] `https://github.com/CTU-IIG/macan`

by the attacker impersonating another device in group authentication, however with no practical consequences. We believe that this is a dangerous mistake to leave in a reference paper, which could lead to flawed implementations if left undetected. We could confirm the attack that allowed forging authenticated signals was present in the implementation. With it an attacker could forge potentially dangerous authenticated messages from legitimate ones. Comparing the models with the implementation also helped us to reveal some minor bugs that were introduced when coding it, and would have probably not been revealed by simple testing.

4.3 Formal Analysis of CANAuth

In this section we analyse the CANAuth protocol, which we presented in Section 2.3.1. CANAuth sends counters as part of the signature for a message, instead of relying on timestamps. A receiving ECU accepts a message only if the counter is higher than any other counter value previously observed, ensuring replay protection but requires greater bandwidth.

We first present our modelling in ProVerif for the two phases of the protocol, key establishment and message authentication. We show to what extent we are able to prove the security in ProVerif.

4.3.1 Key Establishment

In Figure 4.7 we model the key establishment phase of the protocol with ProVerif and we try to verify key secrecy in this model. The system *Sys* defines a pre-shared key kp and a term *error* not known to the attacker, then runs ECU_i and ECU_j.

The procedure starts with ECU_i, which first generates a fresh counter *cnt* and a fresh random number *rnd*. Then it computes the session key *ks* as the *hmac* signature, using the preshared key kp, of the two values *cnt* and *rnd*, and the signature s_2 of the same values using the newly computed session key *ks*. It emits the event $begin(ks)$ signalling the start of the authentication process, and finally sends the challenge message $\langle cnt, rnd \rangle$ and the signature $hmac(ks, \langle cnt, rnd \rangle)$ over the public channel.

At the other end the process ECU_j receives the counter *cnt* and random value *rnd*. It internally computes the session key *ks* knowing kp and its version of the signature s_1, and the transmitted information *cnt* and *rnd*, then receives a signature s_2. If s_1 and s_2 match ECU_j emits an event $end(ks)$, signalling successful completion of the

$$
\begin{aligned}
ECU_i \triangleq & \textbf{new } cnt : bitstring; \\
& \textbf{new } rnd : bitstring; \\
& \textbf{let } ks = hmac(kp, \langle cnt, rnd \rangle) \textbf{ in} \\
& \textbf{let } s_2 = hmac(ks, \langle cnt, rnd \rangle) \textbf{ in} \\
& \textbf{event } begin(ks); \\
& \textbf{out}(c, \langle cnt, rnd \rangle); \\
& \textbf{out}(c, s_2); \\
& \textbf{in}(c, =ks); \\
& \textbf{out}(c, error); 0
\end{aligned}
$$

$$
\begin{aligned}
ECU_j \triangleq & \textbf{in}(c, \langle cnt, rnd \rangle : \langle bitstring, bitstring \rangle); \\
& \textbf{let } ks = hmac(kp, \langle cnt, rnd \rangle) \textbf{ in} \\
& \textbf{let } s_1 = hmac(ks, \langle cnt, rnd \rangle) \textbf{ in} \\
& \textbf{in}(c, s_2 : signature); \\
& \textbf{if } s_1 = s_2 \textbf{ then} (\\
& \quad \textbf{event } end(ks); \\
& \quad \textbf{out}(c, confirm); 0 \\
&) \textbf{ else out}(c, reject); 0
\end{aligned}
$$

$$
Sys \triangleq \textbf{new } error : bitstring; \textbf{new } kp : key; (!\, ECU_i \,|\, !\, ECU_j)
$$

Figure 4.7: CANAuth Key Establishment

authentication phase, and sends back a positive acknowledgement, otherwise it rejects the signatures signaling an error.

We want to know if the attacker is able to find the pre-shared key kp or the session key ks. To check whether ks ever gets revealed, ECU_i listens for ks on the shared channel, and reveals the secret $error$ to represent that the session key has been compromised.

We mark the start and the conclusion with a *begin* and an *end* event, respectively, to know if the authentication property that we defined in Section 2.3 holds for this protocol. Therefore, we check the correspondence between $end(ks)$ and $begin(ks)$.

In these models we encode concatenated messages as tuples. We also discard all modulo operators present in the protocol description: the necessity to cut the length of a number with a modulo operation is due to speed considerations for the session key ks and to space consideration for the signatures sig_A and sig_M, but our symbolic representation of

the protocol does not take these details into account.

Replay attacks that would attempt to establish previously broken session keys are avoided by the use of counters and random values. The protocol should refuse a message containing an old counter, but we cannot express this behaviour in the Applied Pi-calculus in a form that is suitable for verification, as it requires encoding persistent state. Section 6.2 shows how to verify this property using Set-Pi. In our model it will always be possible to re-establish a previous session, so we cannot prove injective agreement on the authentication between ECU_i and ECU_j.

Analysis results ProVerif can prove secrecy of the long term key kp and the session key ks. Also, we can prove weak agreement between the $end(ks)$ and $begin(ks)$: an attacker cannot establish a session key that was not sent previously in the protocol execution.

However, injective correspondence between the two events does not hold in the model: an attacker can always replay the same combination of counter, random value and signature as received by ECU_i and complete authentication with ECU_j. This is due to the same overapproximation that we hit in Section 4.2, and we will show in Chapter 6 how to overcome these obstacles.

4.3.2 Message Authentication

The second phase of CANAuth, message authentication, is modelled on Figure 4.8. Here we assume that a session key ks has already been established, and present only the processes responsible for authenticating messages. The combination of the two phases can be obtained by inserting ECU_i and ECU_j from Figure 4.8 as continuations of the corresponding processes in Figure 4.7.

When ECU_i wants to send a message msg, it generates a new counter cnt and computes the signature $hmac(ks, \langle cnt, msg \rangle)$. ECU_i signals the beginning of the protocol by a *begin* event and then outputs the counter, the message and the signature.

On the other end ECU_j receives the signed message, computes its own version of the signature, and if the received signature matches with its own computation then sends a positive acknowledgement, otherwise it raises the error flag, symbolised by the message $reject$.

$$
\begin{aligned}
ECU_i \triangleq & \textbf{new } msg : signal; \\
& \textbf{new } cnt : bitstring; \\
& \textbf{let } sig = hmac(ks, \langle cnt, msg \rangle) \textbf{ in} \\
& \textbf{event } begin(cnt, msg, sig); \\
& \textbf{out}(c, \langle cnt, msg, sig \rangle); 0 \\
\\
ECU_j \triangleq & \textbf{in}(c, \langle cnt, msg, sig \rangle : \langle bitstring, signal, signature \rangle); \\
& \textbf{let } sig' = hmac(ks, cnt, msg) \textbf{ in} \\
& \textbf{if } sig = sig' \textbf{ then} (\\
& \quad \textbf{out}(c, confirm); \\
& \quad \textbf{event } end(cnt, msg, sig'); \\
&) \textbf{ else out}(c, reject); 0
\end{aligned}
$$

Figure 4.8: CANAuth Message Authentication

Analysis results ProVerif can prove the preservation of secrecy for the long term key kp and the session key ks, which are maintained when the processes of Figure 4.8 are attached to those of Figure 4.7. ProVerif can also prove the weak agreement property between $end(cnt, msg, sig')$ and $begin(cnt, msg, sig)$, but not the strong agreement.

4.4 Discussion

We developed our models of MaCAN and CANAuth using the Applied Pi-calculus variant of ProVerif. We had to change some aspects of the protocols, as described in Section 4.2 and Section 4.3, to be able to analyse them. One such aspect is the use of counters and timestamps to ensure freshness of messages. We modeled both mechanisms as fresh names in Applied Pi-calculus, thus losing their structure.

During our experiments we tried to represent timestamps and counters in two ways: as integers, with an initial constant value $z[]$ and the successor constructor $succ(x)$, where a predicate $\mathsf{lt}(x, y)$ relation would enable comparison between timestamps/counters. Another approach that we tried was to represent them as lists of known values, with an empty list $nil[]$ and a constructor $cons(x, y)$ that appends one element to the list, where a membership predicate $\mathsf{mem}(x, y)$ would indicate whether an element has been seen by the process. Unfortunately none of these approaches works, as the saturation of the predicates $\mathsf{mem}(x, y)$ and $\mathsf{lt}(x, y)$ leads to non-termination issues.

Other analysers such as StatVerif [ARR11] allow the analysis of Applied Pi-calculus processes extended with stateful constructs, but in order to represent potentially infinite timestamps one needs more powerful abstractions that avoid exploring an infinite state-space. Explicitly inserting a fresh timestamp into a list and checking whether the current timestamp is in the list, as we discussed, would generate terms of continuously increasing size, hanging the engine. The same behaviour we encountered in ProVerif, not surprisingly, as they share the same resolution strategy. SAPIC [KK14], an extension of the Applied Pi-calculus using multiset rewriting semantics and the Tamarin prover [SMCB13], became available while this study was being published.

In Chapter 6 we show how we overcome this problem by abstracting names of the Applied Pi-calculus into a finite structure that tracks enough state information to encode the properties of our interest. Appendix B.3 presents Set-Pi models that verify the protocols presented in this chapter.

4.5 Conclusions

This chapter presents an analysis of the two protocols MaCAN and CANAuth using the ProVerif protocol verifier. By this analysis we found a flaw in the key establishment procedure of MaCAN, experimentally verified its presence of an attack in the implementation, and proposed a modified version of the protocol that is immune from the problems we discovered.

Resource constrained networks such as the CAN bus put a strong limit on the design of an authentication protocol. The designers of MaCAN had to rely of custom schemes when designing its procedures, as previous literature did not consider such extreme bounds in terms of bandwidth as 8 bytes of payload per message. Similarly CANAuth relies on a compatible extension (CAN+) and makes careful considerations on the available bandwidth, also avoiding the challenge-response mechanism. We contribute to the protocols with a formal analysis (and an experimental evaluation in the case of MaCAN) and propose changes that fix the flaws we discovered.

The next chapter will study how MaCAN and CANAuth perform in terms of schedulability. We will discover that MaCAN still violates one of the schedulability assumptions in its time distribution procedure, and propose an alternative version of the procedure that prevents a potential Denial of Service attack.

During our analysis we also encountered some limitations in expressing the particular features of MaCAN with the languages and tools of our choice. Chapter 6 presents an extension of the Applied Pi-calculus that deal with these limitations.

Finally, protocols like MaCAN rely on relatively weak cryptography, so we would like to extend our analysis to cover possible attacks in the computational model, and be able to precisely evaluate the level of security of MaCAN.

CHAPTER 5

Schedulability Analysis for Authenticated CAN Protocols

Authenticated CAN protocols increase the security of a car, protecting the internal network from message forgeries and replays. However, the added functionality comes at a cost of computation and bandwidth resources, so we need to determine how these extensions can coexist with the real-time safety features of a microcontroller with limited processing power.

The designs of MaCAN and CANAuth have avoided the traditional challenge-response mechanisms by introducing some state information, in forms of a timestamp or a counter. This choice was necessary because a challenge-response mechanism would double the number of messages sent through the network, and would also require a more complex implementation in systems with tight control loops (where the same type of message is sent at a fixed interval, e.g. one every 50 ms).

The performance impact of challenge-response could make a system unschedulable, therefore all the authenticated extensions to CAN which meet the necessary safety criteria avoid such pattern in favour of the use of more stateful mechanisms, such as counters and timestamps. However, even when using counters or timestamps to reduce the communication overhead, an increased cost in terms of computation time and the

relative added delay are still present.

In this chapter we present the traditional schedulability analysis of CAN as described in [DBBL07], and extend it to take into account the timing impact in computing the cryptographic functions for MaCAN and CANAuth. In analysing the schedulability of MaCAN, we discover a potential Denial of Service attack in the time distribution procedure, and propose an alternative procedure that avoids the issue.

Section 5.1 presents the standard analysis of [THW94, DBBL07] for the schedulability of CAN networks. Sections 5.3 and 5.4 extend the analysis to take into account the features of MaCAN and CANAuth, respectively. Section 5.5 analyses the impact of the time-triggered extensions on schedulability with authentication. Finally, we conclude in Section 5.6.

5.1 Review: Schedulability Analysis

For the purpose of schedulability, a CAN bus network can be seen as a *real-time, non-preemptive, fixed-priority system*. A *real-time* system has a set of n independent tasks (signals in the case of CAN bus) that need to be scheduled for execution (transmission). Each task i executes at a recurring period $T_i \in \mathbb{R}^+$, requires $C_i \in \mathbb{R}^+$ time to complete, and has an associated deadline $D_i \in \mathbb{R}^+$. In the case of a CAN bus network there is a set of messages, and each message m_i needs to be sent periodically with period T_i, occupies the communication channel for the required transmission time C_m, has a fixed deadline D_m and a queuing jitter J_m associated.

CAN bus is a *non-preemptive* system: after arbitration the transmitting ECU occupies the channel until the transmission is completed, and the current transmission cannot be interrupted for rescheduling. CAN bus has also a *fixed-priority* between messages: there is a total priority order given by the message IDs. A message m_i with ID i has higher priority than m_j with ID j if and only if $i < j$.

The system is schedulable if the computed worst case response time R_m for each message meets its respective deadline D_m: hence for all message m we must have $R_m \leq D_m$.

This section presents the worst case schedulability analysis that was first elaborated in [THW94] and then corrected in [DBBL07, BLV07].

Given a message m, the transmission time τ_{bit} for one bit and the payload length in bytes s_m, we can compute the transmission time for the message as follows:

$$C_m = \left(g + 8s_m + 13 + \left\lfloor \frac{g + 8s_m - 1}{4} \right\rfloor \right) \tau_{bit} \tag{5.1}$$

The term under the floor operation takes into account *bit stuffing*: this mechanism inserts a bit of opposite polarity for each sequence of 5 bits of the same polarity, to ensure synchronisation.

The value of g represents the number of bits used for the fixed-length fields of the frame. It is set to 34 for the standard 11-bit identifier format, and to 54 for the extended 29-bit identifier format. We can therefore simplify (5.1) to (5.2) for 11-bit identifiers and (5.3) for 29-bit identifiers.

$$C_m = (55 + 10s_m)\tau_{bit} \tag{5.2}$$

$$C_m = (80 + 10s_m)\tau_{bit} \tag{5.3}$$

DEFINITION 1 (WORST CASE RESPONSE TIME) *The worst case response time R_m as defined by [THW94] in their original analysis is defined by the formula:*

$$R_m = J_m + w_m + C_m \tag{5.4}$$

where J_m indicates the queuing jitter, which is the longest time between the initiating event of a message and the message being queued, and w_m is the worst case delay on the channel before message m can be transmitted.

We are now going to detail how to compute the worst case response time.

Since CAN is a non-preemptive system, the channel might not be available for transmission. We define this time window as the maximum blocking time before arbitration B_m. The maximum blocking time is equal to the maximum transmission time of lower priority messages that might occupy the channel when message m is queued:

$$B_m = \max_{k \in lp(m)} (C_k) \tag{5.5}$$

The queuing delay is defined by the following recursive formula:

$$w_m^0 = B_m \tag{5.6}$$

$$w_m^{n+1} = B_m + \sum_{k \in hp(m)} \left\lceil \frac{w_m^n + J_k + \tau_{bit}}{T_k} \right\rceil C_k \tag{5.7}$$

Where $hp(m)$ denotes the messages with priority higher than m.

The definition of w_m^n is monotonic on the parameter n, therefore it defines a fix-point computation that allows us to find the worst case response time for message m.

The analysis of the system terminates when either:

1. $\exists m, i \,.\, J_m + w_m^i + C_m > D_m$, the worst case response time for message m at iteration i already exceeds its deadline, in which case the system is not schedulable;
2. $\forall m \exists i \,.\, w_m^i = w_m^{i+1} \wedge R_m \leq D_m$, in which case the fix-point is reached and the system is schedulable with the computed worst case response times.

Before discussing the correctness of this analysis, we define the concept of level-m busy period.

DEFINITION 2 (LEVEL-m BUSY PERIOD) *In non-preemptive fixed priority scheduling,* level-m busy period *is a period $[t^s, t^e)$, where t^s is the time when the first message of priority level m or higher is scheduled for being transmitted, and t^e is the corresponding ending time, when all messages of priority level m or higher have already been transmitted, therefore no messages of priority m or higher in the current busy period can interfere with any future enqueuing of a level-m message.*

Revised analysis The original analysis by Tindell et al. [THW94] assumes that the deadline for each message is bounded by its period ($D_m \leq T_m$) to enforce a queue of length at most 1. It does not take into account that with non-preemptive scheduling the level-m busy period might exceed the time at which the next instance of message m is queued. Even then, it is not always the case that $D_m \leq T_m$, as in some applications the deadline needs to be set above the period [DBBL07].

If $t_m \leq T_m - J_m$ then the busy period ends before the successive message is queued, so the standard analysis is still valid. Otherwise $t_m > T_m - J_m$ and there can be more than one level-m message queued for transmission at a given time. In this case we can expect two different behaviours: new messages are queued for transmission, or new messages will overwrite previous ones. The analysis just presented is still fine for the overwriting behaviour, but does not consider that more than one message with the same ID m may be queued for communication at any given time.

Davis et al. [DBBL07] proposed a revised analysis to compute the correct worst case response times when $t_m > T_m - J_m$. They only consider the queuing behaviour as it is the one that [THW94] does not.

The level-m busy period can be computed as the fix-point of:

$$t_m^0 = C_m \tag{5.8}$$

$$t_m^{n+1} = B_m + \sum_{k \in hp(m)} \left\lceil \frac{t_m^n + J_k}{T_k} \right\rceil C_k \tag{5.9}$$

this fix-point computation above is guaranteed to be converging if the bus utilisation is less than 1.

The bus utilisation is defined as:

$$U_m = \sum_{k \in hep(m)} \frac{C_k}{T_k} \tag{5.10}$$

where $hep(m)$ denotes all messages with higher or equal priority to m.

Assuming that messages are queued in the network, the number of messages that are ready before the end of the busy period is:

$$Q_m = \left\lceil \frac{t_m + J_m}{T_m} \right\rceil \tag{5.11}$$

this number corresponds to the maximum queue size for message m, and takes into consideration both the level-m busy period and the worst case jitter necessary to insert a message in queue.

We can now re-define the worst case queuing delay — when q messages of level m are queued for transmission — with the following fix-point computation:

$$w_m^0(q) = B_m + q \cdot C_m \tag{5.12}$$

$$w_m^{n+1}(q) = B_m + q \cdot C_m + \sum_{k \in hp(m)} \left\lceil \frac{w_m^n(q) + J_k + \tau_{bit}}{T_k} \right\rceil C_k \tag{5.13}$$

where the factor $q \cdot C_m$ is added to (5.6, 5.7) to take into account the transmission of all q messages queued to be sent.

The worst case response time for the q-th message in queue is:

$$R_m(q) = J_m + w_m(q) - qT_m + C_m \tag{5.14}$$

because its relative busy period starts at $qT_m - J_m$.

The worst case response time for message m then becomes the maximum response time for all messages in queue:

$$R_m = \max_{q=0\ldots Q_m-1} (R_m(q)) \tag{5.15}$$

Hence schedulability can be computed as before, by substituting the new definitions of $w_m^n(q)$ and R_m to the decision procedure.

5.2 Authenticating Messages

In this section we introduce the common assumptions and definitions that form the basis for our extended analysis for both MaCAN and CANAuth.

Assumptions Key establishment happens at the beginning of a session when the car is started, so this phase of communication can be excluded from the schedulability analysis. It is worth noting that both protocols under our consideration have a maximum life for session keys of 48 hours.

The weak cryptography used by CANAuth and MaCAN would allow an attacker to discover the keys if he observed more messages than those observable in 48 hours of communication [HSV11, HRS12]. This limit on the operational time goes far beyond the normal operational time of a vehicle, hence it is safe to assume that key establishment only happens at startup, when there are no timing requirements.

Definitions Here we present common measures that will be used in both analyses:

1. H_m is the time required for computing the cryptographic primitive for message m;
2. $T^{eof} = 30 \cdot \tau_{bit}$ over-approximates the time taken to transmit the end of the frame, comprising the CRC codes, error flags and the end of frame delimiter;
3. $T_m^{payload} = 8 \cdot s_m \cdot \tau_{bit} + \lceil \frac{8 \cdot s_m \cdot \tau_{bit}}{4} \rceil$ over-approximates the time taken to transmit the payload of message m, considering bit-stuffing.

MaCAN and CANAuth both propose an authentication scheme for signals which, at a high level, requires computing a cryptographic primitive by the sender, recomputing such primitive by the receiver, and check if there is a match. At a closer look, these two protocols have different timing constraints, as we will now discuss.

In Section 5.3 and Section 5.4 we adapt the analysis to the CANAuth and the MaCAN protocols, respectively. Finally, in Section 5.5 we discuss how the TTCAN extension can positively affect the schedulability of MaCAN.

5.3 Schedulability with MaCAN

5.3.1 Message Authentication

Recalling Section 2.3.2, the protocol sends authenticated messages by first performing a request for authentication (Figure 2.6 (a)). The response to this request varies depending on the size of the signal s_m: if $s_m \leq 4$ then the format of Figure 2.6 (c) is used, otherwise a longer signal is split in multiple messages, each containing 2 bytes of actual payload.

Furthermore, the prescaler field from the request for authentication (Figure 2.6 (a)) specifies the frequency of authenticated signals. In this analysis we are going to assume that prescaler is always 1 for authenticated messages, hence every message is authenticated. This will lead to an analysis that is overly pessimistic, especially in the case of signals with length greater than 4, which need to be split into multiple messages. The number of MaCAN messages in which a payload of length s_m will be split is defined as:

$$Split_m = \begin{cases} \lceil \frac{s_m}{2} \rceil & \text{if } m \text{ is authenticated} \\ 1 & \text{otherwise} \end{cases}$$

The MaCAN Crypt Frame is always 8 bytes in size, hence we define a normalisation of the message size:

$$\hat{s}_m = \begin{cases} 8 & \text{if } m \text{ is authenticated} \\ s_m & otherwise \end{cases}$$

The transmission time needs to be redefined by substituting s_m with $\hat{s}_m$:

$$C_m = \left(g + 8\hat{s}_m + 13 + \left\lfloor \frac{g + 8\hat{s}_m - 1}{4} \right\rfloor \right) \tau_{bit} \tag{5.16}$$

The formula for the maximum blocking time due to lower priority messages occupying the channel remains unchanged:

$$B_m = \max_{k \in lp(m)} (C_k) \tag{5.17}$$

However, we redefine the level-m busy period, to take into account the possibility of queuing multiple messages when using signatures, as follows:

$$t_m^0 = C_m \cdot Split_m \tag{5.18}$$

$$t_m^{n+1} = B_m + \sum_{k \in hp(m)} \left\lceil \frac{t_m^n + J_k}{T_k} \right\rceil \cdot C_k \cdot Split_k \tag{5.19}$$

And similarly the bus utilisation is multiplied by the number of messages transmitted at each period:

$$U_m = \sum_{k \in hep(m)} \frac{C_k \cdot Split_k}{T_k} \tag{5.20}$$

$$\tag{5.21}$$

In MaCAN messages are signed with the current timestamp before they are ready to be transmitted; the timestamp, however, is not transmitted along with the message, hence

the receiving end must check the signature against all the timestamps that are valid in the current transmission window.

Let T_{clock} be the rate at which the timestamp is increased, configurable at design time. Depending on T_{clock} and on the deadline D_m, there can be more than one valid timestamp to sign message m.

We denote the number of valid timestamps for a signature of a message m as:

$$N_m = \left\lfloor \frac{D_m}{T_{clock}} \right\rfloor + 1 \tag{5.22}$$

The queue length is increased by the additional computation time H_m required before a message is available for transmission:

$$Q_m = \left\lceil \frac{t_m + J_m + H_m}{T_m} \right\rceil \cdot Split_m \tag{5.23}$$

The worst case delay for the q-th message m in the queue remains the one defined in (5.12, 5.13), since this fix-point computation is only influenced by the use of the channel by higher priority messages, which does not change in MaCAN.

Finally we define the worst case response time for the q-th message m in queue as:

$$R_m(q) = J_m + H_m + w_m(q) - q \cdot T_m + C_m + \max(0, N_m \cdot H_m - T^{eof}) \tag{5.24}$$

this formula takes into account the necessary time for computing the cryptographic primitive by the sending ECU, and that after having received the message payload, the receiving ECU needs to perform at most N_m signature computations before accepting or discarding a message. Such computation is parallel to the transmission of messages in the channel, and it can start just after having received the message payload. The max operator reflects this parallelism in the worst case analysis.

Remarks In MaCAN, securing the system might require finding the right balance between the cost of the cryptographic primitive H_m and the frequency of clock updates T_{clock}: increasing security and thus the cost H_m reduces the potential for system schedulability, while increasing the clock update rate T_{clock} may render the system unschedulable.

On the other hand, decreasing the level of security provided by the signing function might increase the chances for an attacker of finding the session key, or forging a message that passes the signature checks, and decreasing the rate T_{clock} might introduce potential replay attacks within the validity time frame of the signature.

Correctness of the extended analysis The work in [BLV07] presents full proofs of correctness for fixed-priority scheduling with deferred preemption, of which the schedulability of the CAN protocol is a specific case. Inspecting the proofs presented there, we found no assumption that is compromised for this extended set of formulas that capture the behaviour of the MaCAN protocol.

The formulas re-defined in this section touch the computation leading of the maximum queue length, the initial delay of a message and the time used to validate a message upon reception. The initial delay now defined as $J_m + H_m$ can fit the concept of phasing ϕ_m presented in [BLV07] and only influences the result of the analysis, but maintains the validity of the proofs. The time needed to check a message signature ($\max(0, N_m \cdot H_m - T^{eof})$) does not alter any other condition for schedulability than that of the final worst case response time for message m. The rest of the concepts are preserved equal as in Section 5.1.

5.3.2 Time Server

Recalling Section 2.3.2, the time server is responsible for delivering the current timestamp in a MaCAN network, synchronising the ECUs. In order to do this, the time server sends an unauthenticated signal at each update of the timestamp, so that the client ECUs can check whether its internal version is synchronised with the time server.

A non-synchronised ECU would request an authenticated version of the timestamp, by sending a challenge as in message format (2.18), to which the time server would reply in message format (2.19) with the timestamp signed with the shared key between the ECU and the key server. This behaviour represents a problem for the system schedulability, as it violates the requirement of no dependency between tasks. Hence, these messages cannot be captured by the rules that we introduced for the rest of the protocol.

In fact, the dependency introduced by this portion of the protocol constitutes a more serious Denial of Service attack. An attacker willing to compromise the system availability can simply inject an unauthenticated out-of-sync timestamp. At this point all other ECUs believe that their internal clock is not synchronised with the time server, and request an authenticated version of the timestamp by sending a challenge.

A hardware safety mechanism provided by the CAN bus network prevents devices from flooding the network due to software failures, hence a CAN device has a limited number of messages that can be sent over a predefined interval. In this case the mechanism, which also prevents an attacker from simply flooding the network, would slow down the response of the time server for sending authenticated timestamps. This behaviour effectively disables communication until all the other ECUs have received an authenticated timestamp.

Revised timestamp distribution We propose the following alternative procedure for distributing authenticated timestamps. The time server TS generates a key pair pk_{TS}, sk_{TS} at each new session, and publishes pk_{TS} to the key server KS at the end of the key-establishment procedure. The key server KS is responsible for delivering the public key pk_{TS} for each new session established between clients, by including it at the end of the key establishment payload. Finally, the key server delivers each new version of the timestamp T both in cleartext and encrypted with sk_{TS}:

$$TS \rightarrow ECUs : T, aenc(sk_{TS}, T)$$

In this protocol, the receiving ECUs only need to check that the encryption $aenc(sk_{TS}, T)$ corresponds to the plaintext T if their internal clock is skewed, saving precious computational time. Replay attacks are detected and ignored by refusing old timestamps, and newer timestamps cannot be forged by the attacker unless it learns sk_{TS}. The correctness of this approach is shown with a Set-Pi model, in Appendix B.3.

5.4 Schedulability with CANAuth

In CANAuth instead, each authenticated message needs to be sent with the highest counter for its authentication group. This means that if another ECU is sending an authenticated message on the same group, the signature computed for a message in queue is invalidated. Therefore the time available for computing the signature needs to fit in the period between the end of transmission of the previous message payload and the end of transmission of the current message ID. If the signature computation does not fit within this period of time, then we need to make sure that there is an available slot for communicating the message with a valid signature, before a new message from the same group arrives.

Since a message in the same authentication group can invalidate the current signature, we need to define the authentication delay as the time needed to re-compute the signature:

$$A_m = \max(0, H_m - T_m^{payload} - T^{eof}) \tag{5.25}$$

here we remove $T_m^{payload}$ and T^{eof} from the delay because an ECU can assume that its signature has been invalidated when it reads on the channel the identifier of a message in the same authentication group.

The worst case delay for the q-th message m then becomes:

$$w_m^0(q) = B_m \tag{5.26}$$

$$w_m^{n+1}(q) = B_m + q \cdot C_m + \sum_{k \in hp(m)} \left\lceil \frac{w_m^n(q) + J_k + \tau_{bit}}{T_k} \right\rceil C_k + A_m \tag{5.27}$$

note that this formula assumes the continuity of the level-m busy period. This implies that the transmission of q consecutive messages with identifier m must be performed without invalidating the signatures, and therefore introducing delays that could be filled by lower priority messages.

If this condition cannot be satisfied, then it would be impossible to guarantee continuity of the busy period, and the analysis would be invalidated. For example when $H_m > C_m$, it is impossible to transmit a sequence of consecutive messages with the same identifier, and continuity of the busy period is violated.

Finally we adapt the worst case response time for the q-th level-m message as follows:

$$R_m(q) = J_m + w_m(q) - q \cdot T_m + \max(C_m - T^{eof} + H_m, C_m) \tag{5.28}$$

here we impose that a message is available to the receiver only when the signature has been checked.

Remarks In CANAuth, the choice of cryptographic primitive can have a strong influence in the schedulability analysis. We noted that if the computation cost H_m is higher than the time needed for transmitting the message C_m, and if more than one level-m message can be enqueued at any time, then the level-m busy period can be interleaved with lower priority messages and therefore the analysis is invalidated.

Even if the continuity of the busy period is satisfied, there worst-case delay can still increase if a previous message invalidates the signature. This places a strict limit on the strength of the cryptographic primitive.

Correctness of the extended analysis Similarly to the argument for the correctness of the analysis for MaCAN in Section 5.3, we argue that the extended analysis does not alter the proofs of correctness presented in [BLV07], with one important exception: the invalidation of a signature for message m might break the level-m busy period while messages of priority m or higher are scheduled for transmission, but have invalid signature that need recomputing. In that case, a message m' with lower priority can successfully win arbitration and block the channel and the level-m busy period with a low priority message.

One possible way to avoid this problem is to limit the signature computation time to $T_m^{payload} + T^{eof}$, so that even in case of a signature invalidation the sender is able to recompute a new signature before the next transmission.

5.5 Effects of TTCAN

TTCAN can positively affect the schedulability of a system. Periodic messages are assigned a fixed slot in the system matrix. Arbitrated messages do not present a periodic behaviour, therefore are not subjected to deadlines. Because of the fixed scheduling in TTCAN, response time are defined statically, hence there is no need for a worst-case analysis.

When using the MaCAN protocol on top of TTCAN, the fixed priority scheduling defined in terms of basic cycles and system matrices also defines what are the constraints for the schedulability of MaCAN:

1. for each message m under fixed priority scheduling, scheduled to be transmitted at time t in the current cycle, the payload information needs to be available at least at time $t - H_m$ for the signature computation;

2. on the receiving side messages need to be checked for authentication. Unlike in the worst-case schedulability analysis, the timestamp used for authentication is identified by the fixed priority scheduling. Only one combination of message and timestamp then needs to be checked for authentication on the receiving side, therefore they will be available at time $t + C_m + H_m$;

3. differently from standard CAN, TTCAN incorporates a notion of synchronised time that is transmitted with the reference message at the start of each basic cycle, possibly eliminating the need for an additional time-server.

When using the CANAuth protocol on top of TTCAN, we can recognise the benefit that each message m under fixed priority scheduling, scheduled for transmission at time t, has statically known time distances to other messages in a group. It is therefore easy to compute the counter value that will be used to sign message m at time t, or to statically enforce that there is enough time for the sending ECU to compute the right signature after another message in the same group of m has been sent.

5.6 Conclusions

In this chapter we have seen how the schedulability analysis for the standard CAN protocol can be adapted to take into account the authenticated extensions. As it is common in many situations where we need to guarantee both safety and security properties in a safety critical system, the two sets of properties are usually conflicting,

and we need to find a balance between the level of security that we put in the system and the impact over its safety characteristics, like real-time schedulability.

In particular the MaCAN and CANAuth protocols avoided the challenge response scheme to ensure freshness, as their authors deemed it too costly for the application. They also resorted to the use of timestamps and counters, respectively, and accepted the costs of that choice, resulting in more complex protocols.

This study on their schedulability shows that MaCAN degrades its performances with the increasing of precision in its timestamp mechanism. The authenticated time distribution procedure of MaCAN contains a dangerous dependency between messages that allows an attacker to take down the system by sending skewed unauthenticated timestamps. CANAuth on the other side is limited in the time available for signing the messages, but its design leads to simpler schedulability results.

The Time-Triggered CAN protocol — which is designed as a safety feature — positively affects both MaCAN and CANAuth because of the added restrictions in the communication.

CHAPTER 6

Extending the Applied Pi-calculus with Sets

As we have seen throughout this thesis, communication protocols often rely on stateful mechanisms to ensure certain security properties. In our case study counters and timestamps are used to ensure authentication. In other situations the security of communication can depend on whether a particular key is registered to a server or it has been revoked.

In Chapter 3 we studied how ProVerif, like other state of the art tools for protocol analysis, achieves good performance by converting a formal protocol specification into a set of Horn clauses. These Horn clauses represent a monotonically growing set of facts that a Dolev-Yao attacker can derive from the system. Since this set of facts is not state-dependent, the category of protocols of our interest cannot be precisely analysed by such tools, as they would report false attacks due to the over-approximation.

In this chapter we present Set-Pi, an extension of the Applied π-calculus that includes primitives for handling databases of objects, and propose a translation from Set-Pi into Horn clauses that employs the set-membership abstraction to capture the non-monotonicity of the state. Furthermore, we give a characterisation of authentication properties in terms of the set properties in the language, and prove the correctness of our approach. Finally we showcase our method with two examples, a simple authentication protocol based on counters, and key registration protocol.

6.1 Introduction

The automated verification of security protocols has been the subject of intensive study for about two decades now. This has resulted in methods and tools that are feasible for finding attacks or proving the absence of attacks for a large class of protocols. One of the most successful approaches is static analysis, as for instance used in the ProVerif tool [Bla01, Bla05, Wei99, GL08, BBD+05]. The key idea of this approach is to avoid the exploration of the state space of a transition system, but rather compute an over-approximation of the set of messages that the intruder can ever learn. The abstraction is efficient because it avoids the common state-explosion of model checking and it does not require a limitation to finite state-spaces. While this works fine for many protocols, we get trivial "attacks" if a protocol relies on a notion of state that is not local to a single session.

Simplifying the CANAuth protocol from Chapter 2, one can ensure authentication as follows:

$$A \to B \ : \ \{Msg, Counter\}_{Key}$$

where *Key* is a symmetric key known only to A and B, *Msg* is some payload message and *Counter* is the current value of a counter used for avoiding replay attacks: B accepts a message only if *Counter* is strictly greater than in the last accepted message from A. This protocol thus ensures *injective agreement* [Low97] on *Msg*, since B can be sure that A has sent *Msg* and it is not a replay, i.e., even if A chooses to transmit several times the same Payload *Msg*, B will not accept it more often than A sent it. There are of course several ways to model such a counter in the applied π calculus, the input language of ProVerif, however none is going to work in the abstraction due to its *monotonicity*: roughly speaking, whatever B accepts once, he will accept any number of times and we thus get trivial attacks. In fact, verifying injective agreement properties in ProVerif requires a dedicated mechanism [Bla09].

The above message is taken from the CANAuth protocol [HSV11] that is intended for the automotive industry and needs to work under strong limitations on bandwidth and time. Due to these constraints, standard mechanisms like challenge-response (B first sends a nonce, then A includes it in the message instead of the counter) are no option. But even without such bounds there are practical real-world examples that today's abstraction approaches cannot support:

- Key update/revocation: after updating an old key with a fresh one, one does not accept messages encrypted with the old key anymore (at least after some grace period).
- Key tokens/hardware security modules: they maintain a set of keys of different status and attribute, and can be communicated with through an API. When changing the status of a key, an operation may no longer be possible with that key.

- Data bases: an online shop that maintains a database of orders along with their current status; a customer may cancel an order, but only as long as it has not entered the status "shipped".

More generally, systems that have a notion of state (that is not local to a session) and that have a *non-monotonic* behavior — i.e. an action is possible until a certain change of state and that is disabled afterwards are incompatible with the abstraction of tools like ProVerif.

Contributions In this chapter we formally define the novel Set-Pi calculus that extends the popular applied π calculus [AF01] by a notion of sets of messages. It allows us to declaratively specify how processes can store, lookup and manipulate information like sets of keys, orders, or simply counters as in the above example. (Note that this does not increase the expressive power of applied π, since one could also simulate sets using private channels.) The semantics gives rise to an infinite-state transition system since we can model unbounded processes that generate any number of fresh messages. We can define state-based queries for Set-Pi, that ask for attacker-derivable messages, their set membership status, and boolean combinations thereof. A specification is secure iff no query is satisfied in any reachable state. Note that we do not specify a particular attacker, but more generally prove that the protocol is secure in the presence of an *arbitrary* attacker A that can be specified as a Set-Pi calculus (without access to restricted names and sets).

The second contribution is a stateful abstraction for Set-Pi. The idea is that the abstraction of a message incorporates the information to which sets it belongs, and we model how this set membership can change. In doing so, we integrate the essential part of the state information into an otherwise stateless abstraction. This fine balance allows us to combine the benefits of stateless abstraction — namely avoiding state explosion and bounds to finite state spaces — and at the same time support a large class of protocols that rely on some state aspects.

Formally, this abstraction is a translation from a Set-Pi protocol specification and a set of queries into a set of first-order Horn clauses. Our third contribution is to prove a soundness result for this abstraction: every reachable state is abstractly represented by the Horn clauses. In particular, if the Horn clauses have a model, then the given Set-Pi specification is secure for the given queries and against an arbitrary Set-Pi-attacker. For checking whether the Horn clauses have a model, we can use various automated tools like ProVerif.

Finally, we demonstrate the practical feasibility of our approach in our case study of the MaCAN and CANAuth protocols presented in Chapter 4; this chapter will contain only excerpts as illustrating examples, while the complete models of the protocols can be

found in the appendix Section B.3.

The rest of the chapter proceeds as follows: Section 6.2 introduces the language and presents, as a running example, a simplified version of CANAuth. Section 6.3 describes the type system. Section 6.4 gives an instrumented semantics for the language. Section 6.5 presents two definitions for weak and strong authentication and provides a mechanised way to encode such properties in Set-Pi. Section 6.6 present our translation of Set-Pi into Horn clauses and Section 6.7 proves the correctness of this approach. In Sections 6.8 and 6.9 we summarize our results and discuss related work.

6.2 Syntax

The calculus is presented in Figure 6.1. As in ProVerif, we have terms M,N which are either variables, names – annotated with a sequence of terms – or constructor applications. Constructors are generally accompanied by destructors defined as rewrite rules that describe cryptographic primitives. For example:

$$\textbf{reduc}\ \forall x_m : t_m, x_k : key\ .\ sdec(x_k, senc(x_k, x_m)) \rightarrow x_m;$$

models symmetric key encryption: for every message x_m, key x_k, if a process knows an encrypted message $senc(x_k, x_m)$ and the key x_k then it can obtain the message x_m. For any rewrite rule of the form **reduc** $\forall \vec{x} : \vec{T}\ .\ g(M_1, \ldots, M_n) \rightarrow M$; we require that $fv(M) \subseteq fv(M_1, \ldots, M_n) \subseteq \{\vec{x}\}$.

Processes P,Q are: the stuck process 0, replication – which is marked by a label k, parallel composition of two processes, output, typed input, restriction – marked by a label l – and destructor application. We require that processes are closed and that they are properly alpha-renamed. Note that the user does not specify annotated names and labels in the initial process, hence the grey color. Names are introduced by the semantic step for restriction, and unique labels are automatically inserted by the parser.

The distinguishing feature of our calculus is the ability to track values in databases: the membership test (**if** b **then** P **else** Q) allows us to check a membership condition b, while a set transition (**set**(b^+); P) inserts and removes terms from sets, according to b^+. Finally we use locks on sets to ensure linearity of set transitions: the construct **lock**(L) prevents all other processes to modify sets in L in the continuation, while **unlock**(L) releases the locks on L.

A system *Sys* is the context in which the process operates, and declares the available sets and rewrite rules. We mark with $\mathbb{P}$ the set of processes produced by the syntactic category P and with $\mathbb{M}$ the set of terms produced by M. To avoid ambiguity, we

$$
\begin{aligned}
M,N &::= x \mid a^l[M_1,\ldots,M_n] \mid f(M_1,\ldots,M_n) \\
P,Q &::= 0 \mid !^k P \mid P_1 \mid P_2 \\
&\quad \mid \mathbf{out}(M,N);P \mid \mathbf{in}(M,x:T);P \mid \mathbf{new}^l\ x:a;P \\
&\quad \mid \mathbf{let}\ x = g(M_1,\ldots,M_n)\ \mathbf{in}\ P\ \mathbf{else}\ Q \\
&\quad \mid \mathbf{if}\ b\ \mathbf{then}\ P\ \mathbf{else}\ Q \mid \mathbf{set}(b^+);P \\
&\quad \mid \mathbf{lock}(L);P \mid \mathbf{unlock}(L);P \\
b &::= b_1 \wedge b_2 \mid b_1 \vee b_2 \mid \neg b \mid M \in s \\
b^+ &::= b_1^+; b_2^+ \mid M \in s \mid \neg M \in s \\
Sys &::= \mathbf{new}\ s:\mathbf{set}\ T; Sys \\
&\quad \mid \mathbf{reduc}\ \forall \vec{x}:\vec{T}\ .\ g(M_1,\ldots,M_n) \rightarrow M; Sys \mid P
\end{aligned}
$$

Figure 6.1: The process calculus

mark with $\mathbb{S} = \{\mathsf{s}_1,\ldots,\mathsf{s}_n\}$ the sets declared in a specific instance of *Sys*, while we use s,s',s_1,s_2 and so on to denote any of the sets in $\mathbb{S}$.

As syntactic sugar we add the following features to Set-Pi:

- n-tuples $\langle M_1,\ldots,M_n\rangle$, which can be encoded with a constructor $mktpl_n(M_1,\ldots,M_n)$ and n destructors $\mathbf{reduc}\ \forall \vec{x}:\vec{T}\ .\ proj_n^i(mktpl_n(x_1,\ldots,x_n)) \rightarrow x_i$;
- pattern matching on tuples for **let** bindings and inputs, which can be encoded using multiple let bindings with the destructors $proj_n^i$ and equality tests;
- $!^{\{s_1,\ldots,s_n\}}P$ means the replication of P that locks sets $s_1,\ldots,s_n$ before its execution; the semantics releases the locks when P reduces to 0;
- and omitting **else** branches where not needed.

CANAuth example As a running example we use CANAuth, a protocol that runs on top of resource limited CAN bus networks and — due to the real-time requirements of CAN bus networks — uses one way communication from source to destination, avoiding challenge-response patterns. The low level mechanism that is used to ensure freshness properties is the use of counters together with message authentication codes. Comparing a counter with the highest value previously received allows to ensure that a message cannot be replayed.

Here we model a simplified form of its message authentication procedure, that assumes that the two communicating parties, a sender Alice and a receiver Bob, have established a session key k and are both keeping track of their own local copy of a counter c.

In order for Alice to send a message m to Bob, she signs m and her own counter c increased by one with the shared key k. Here we denote with $hmac(msg(c),k)$ such signature. Bob receives the message and checks whether the counter c is already in the set *received*; if not, it accepts the message.

$$
\begin{aligned}
A \triangleq\ & \mathbf{new}^1\ c : cnt; \\
& \mathbf{let}\ m = msg(c)\ \mathbf{in} \\
& \mathbf{event}\ send(m); \\
& \mathbf{out}(ch, \langle m, hmac(m,k)\rangle); 0 \\
\\
B \triangleq\ & \mathbf{in}(ch, \langle x_m, x_s\rangle : \langle msg(cnt), hmac(msg(cnt), key)\rangle); \\
& \mathbf{let}\ x_c = getcnt(x_m)\ \mathbf{in} \\
& \mathbf{let}\ \_ = checksign(x_m, x_s, k)\ \mathbf{in} \\
& \quad \mathbf{if}\ x_c \notin received\ \mathbf{then} \\
& \quad\quad \mathbf{set}(x_c \in received); \\
& \quad\quad \mathbf{event}\ accept(x_m); 0 \\
\\
S \triangleq\ & \mathbf{new}\ received : \mathbf{set}\ cnt; \\
& \mathbf{reduc}\ \forall x : cnt\ .\ getcnt(msg(x)) \rightarrow x; \\
& \mathbf{reduc}\ \forall x : t, k : key\ .\ checksign(x, sign(x,k), k) \rightarrow x; \\
& \mathbf{new}^2\ k : key; \\
& (!^3 A \mid !^{4\ \{received\}} B)
\end{aligned}
$$

In order to express authentication we insert two events: *send* and *accept*. These are just syntactic sugar for set operations and, as we show in Section 6.5, they can be translated into set operations.

6.3 Type system

The type system presented in Figure 6.2 is constructed to track the membership of values in sets. We denote by *Sym* the set of symbols for terms, destructors and sets that occur in a process; the category of types for *Sym* is T^{Sym}.

Data types T are either type variables, name types or constructors over types. Name

$$
\begin{array}{rll}
T & ::= t \mid a \mid f(T_1,\ldots,T_n) & \text{algebraic data types} \\
T^{Sym} & ::= T & \text{data types} \\
& \mid (T_1,\ldots,T_n) \to T & \text{destructor type} \\
& \mid \textbf{set}\ T & \text{set type}
\end{array}
$$

Figure 6.2: Type system

types (a) are atomic types like *key* or *cnt*, type variables (t) are used to make destructors polymorphic, and constructor types define the shape of a constructor. For example the term $pk(skey^I[])$ has type $pkey \triangleq pk(skey)$, and a possible type for the constructor $senc(x_k,x_m)$ could be $senc(key,pair(id,pkey))$, if we give the type $pair(id,pkey)$ to x_m and key to x_k.

Destructor types are of the form $(T_1,\ldots,T_n) \to T$ where we require $fv(T) \subseteq fv(T_1,\ldots,T_n)$. A destructor can therefore be applied to different types of data in the process, as long as the typing judgement instantiates a ground type when it is applied. For example the destructor **reduc** $\forall x_m : t, x_k : key\ .\ sdec(x_k,senc(x_k,x_m)) \to x_m$; has type $(key,senc(key,t)) \to t$, while the instantiation $sdec(x_k,senc(x_k,x_m))$ has type $(key,senc(key,pair(id,pkey))) \to pair(id,pkey)$ considering the previous type assignment.

Set types specify the type of terms contained in sets. For example a set of type **set** $pk(skey)$ contains public keys.

The typing rules (Figure 6.3) enforce the correct typing of processes. Γ is the type environment, a map from identifiers of terms, destructors and sets to their type.

The typing rules for terms check whether the environment contains the right types for variables, and build types accordingly for the constructors. The rules for systems create the type environment required for typing destructors and set operations in processes. The rule for destructors applies the substitution $\sigma = \{^{T_i}/_{x_i}\}$ to the terms $M_1,\ldots,M_n,M$ in order to obtain the type of the destructor, while the rule for sets simply adds the type to the environment.

Processes are typed recursively on their syntactic form. The rules for the stuck process, parallel, replication and output simply try to type the continuation under the same Γ. The rules for input and restriction add to Γ the type of the new bound variables. The rules for **if** and **set** check that the membership test or the set transitions are well-formed (i.e. M is of type T when s is of type **set** T) and that the interested sets are locked. Finally, the rule for **let** infers the type of the result of the destructor application given

Terms:

$$\frac{}{\Gamma \vdash x : T}\ \Gamma(x) = T \qquad \frac{}{\Gamma \vdash a[V] : a} \qquad \frac{\Gamma \vdash M_1 : T_1 \quad \cdots \quad \Gamma \vdash M_n : T_n}{\Gamma \vdash f(M_1,\ldots,M_n) : f(T_1,\ldots,T_n)}$$

Systems:

$$\frac{\Gamma[g \mapsto (\sigma M_1,\ldots,\sigma M_n) \to \sigma M] \vdash Sys}{\Gamma \vdash \mathbf{reduc}\ \forall \vec{x} : \vec{T}\ .\ g(M_1,\ldots,M_n) \to M; Sys}\ \sigma = \{^{T_i}/_{x_i}\}$$

$$\frac{\Gamma[s \mapsto \mathbf{set}\ T] \vdash Sys}{\Gamma \vdash \mathbf{new}\ s : \mathbf{set}\ T; Sys} \qquad \frac{\emptyset, \Gamma \vdash P}{\Gamma \vdash P}$$

Processes:

$$\frac{}{\emptyset, \Gamma \vdash 0} \qquad \frac{\emptyset, \Gamma \vdash P_1 \quad \emptyset, \Gamma \vdash P_2}{\emptyset, \Gamma \vdash P_1 \,|\, P_2} \qquad \frac{\emptyset, \Gamma \vdash P}{\emptyset, \Gamma \vdash !^l P} \qquad \frac{L, \Gamma \vdash P}{L, \Gamma \vdash \mathbf{out}(M,N); P}$$

$$\frac{L, \Gamma[x \mapsto T] \vdash P}{L, \Gamma \vdash \mathbf{in}(M, x : T); P} \qquad \frac{L, \Gamma[x \mapsto a] \vdash P}{L, \Gamma \vdash \mathbf{new}^l\ x : a; P} \qquad \frac{L, \Gamma \vdash b^+ \quad L, \Gamma \vdash P}{L, \Gamma \vdash \mathbf{set}(b^+); P}$$

$$\frac{\Gamma \vdash M_i : \sigma T_i \quad L, \Gamma[x \mapsto \sigma T] \vdash P \quad L, \Gamma \vdash Q}{L, \Gamma \vdash \mathbf{let}\ x = g(M_1,\ldots,M_n)\ \mathbf{in}\ P\ \mathbf{else}\ Q}\ \Gamma(g) = (T_1,\ldots,T_n) \to T$$

$$\frac{L, \Gamma \vdash b \quad L, \Gamma \vdash P \quad L, \Gamma \vdash Q}{L, \Gamma \vdash \mathbf{if}\ b\ \mathbf{then}\ P\ \mathbf{else}\ Q} \qquad \frac{L \cup L', \Gamma \vdash P}{L, \Gamma \vdash \mathbf{lock}(L'); P}\ L' \cap L = \emptyset$$

$$\frac{L \setminus L', \Gamma \vdash P}{L, \Gamma \vdash \mathbf{unlock}(L'); P}\ L' \subseteq L$$

Conditions:

$$\frac{L, \Gamma \vdash M : T}{L, \Gamma \vdash M \in s}\ \Gamma(s) = \mathbf{set}\ T, s \in L \qquad \frac{L, \Gamma \vdash b_1 \quad L, \Gamma \vdash b_2}{L, \Gamma \vdash b_1 \wedge b_2}$$

$$\frac{L, \Gamma \vdash b_1 \quad L, \Gamma \vdash b_2}{L, \Gamma \vdash b_1 \vee b_2} \qquad \frac{L, \Gamma \vdash b_1^+ \quad L, \Gamma \vdash b_2^+}{L, \Gamma \vdash b_1^+; b_2^+} \qquad \frac{L, \Gamma \vdash b}{L, \Gamma \vdash \neg b}$$

Figure 6.3: Typing rules for terms, rewrite rules, processes and boolean expressions.

the types of $M_1, \ldots, M_n$, by finding a type substitution that allows typing all arguments of the destructor and by applying such substitution to the result type.

We allow destructor definitions to contain type variables, while we require processes and sets to have only terms of ground types. In the Section 6.4 we introduce a formal semantics for the language, together with the necessary subject reduction results for the type system.

6.4 Semantics

We define in Figure 6.4 an instrumented operational semantics for the language. We have transitions of the form $\rho, S, \mathscr{P} \to \rho', S' \mathscr{P}'$ where:

- $\rho : \mathsf{Var} \to \mathbb{M}$.
- $S \subseteq \mathbb{S} \times \mathbb{M}$ records the set-membership states,
- $\mathscr{P} \subseteq \mathbb{P} \times \wp(\mathbb{S}) \times \wp(\mathbb{M})$ is a multiset of concurrent processes, which are represented as triplets (P, L, V) where P is a process, L is the set of locks held by P, and V is a list of terms that influenced the process (either session identifiers or inputs).

A configuration $\rho, \mathscr{S}, \mathscr{P}$ represents the parallel execution of all processes in $\mathscr{P}$:

$$\Big|_{(P_i, L_i, V_i) \in P} \rho(P_i)$$

In the semantic rules we assume Γ to contain the type definitions for sets, constructors and destructors, and the initial process to be well-typed according to Γ.

The concrete semantics presented here is a synchronous semantics, which we choose for simplicity and in accordance with the previous related work on ProVerif [Bla09].

The rule NIL removes the process 0 when it holds no locks. The rule COM matches an input and an output processes if the output has the type required by the input. Note that the set V_1 of influencing terms for the input process is increased with the term N' constructed from type T using the function pt_x^V. The purpose of pt_x^V is to substitute any type T with a term N' that is homomorphic to T: that is, for every occurrence of a name a in the type T, it produces a variable $x_{a,V}$ that is syntactically different from all other variable occurrences, and every occurrence of a constructor type produces a constructor term of the same form.

$$\rho, S, \mathscr{P} \uplus \{(0, \emptyset, V)\} \to \rho, S, \mathscr{P} \qquad \text{NIL}$$

$$\rho, S, \mathscr{P} \uplus \{(\mathbf{in}(M, x:T); P_1, L_1, V_1), (\mathbf{out}(M, N); P_2, L_2, V_2)\} \to \qquad \text{COM}$$
$$mgu(N', N) \circ \rho, S, \mathscr{P} \uplus \{(P_1\{^{N'}/_x\}, L_1, N' :: V_1), (P_2, L_2, V_2)\}$$
$$\text{where } \Gamma \vdash N : T \text{ and } N' = pt_x^{ri(V_1)}(T)$$

$$\rho, S, \mathscr{P} \uplus \{(P_1 \mid P_2, \emptyset, V)\} \to \rho, S, \mathscr{P} \uplus \{(P_1, \emptyset, V), (P_2, \emptyset, V)\} \qquad \text{PAR}$$

$$\rho, S, \mathscr{P} \uplus \{(!^k P, \emptyset, V)\} \to \rho\{^k/_{x_k}\}, S, \mathscr{P} \uplus \{(P, \emptyset, x_k :: V), (!^{k+1} P, \emptyset, V)\} \qquad \text{REPL}$$

$$\rho, S, \mathscr{P} \uplus \{(\mathbf{new}^l\ x : a; P, L, V)\} \to \rho, S, \mathscr{P} \uplus \{(P\{^{a^l[V]}/_x\}, L, V)\} \qquad \text{NEW}$$

$$\rho, S, \mathscr{P} \uplus \{(\mathbf{let}\ x = g(M_1, \ldots, M_n)\ \mathbf{in}\ P_1\ \mathbf{else}\ P_2, L, V)\} \to$$
$$\rho, S, \mathscr{P} \uplus \{(P_1\{^M/_x\}, L, V)\} \qquad \text{if } g(M_1, \ldots, M_n) \to_\rho M \qquad \text{LET1}$$

$$\rho, S, \mathscr{P} \uplus \{(\mathbf{let}\ x = g(M_1, \ldots, M_n)\ \mathbf{in}\ P_1\ \mathbf{else}\ P_2, L, V)\} \to$$
$$\rho, S, \mathscr{P} \uplus \{(P_2, L, V)\} \qquad \text{if } g(M_1, \ldots, M_n) \not\to_\rho \qquad \text{LET2}$$

$$\rho, S, \mathscr{P} \uplus \{(\mathbf{if}\ b\ \mathbf{then}\ P_1\ \mathbf{else}\ P_2, L, V)\} \to \rho, S, \mathscr{P} \uplus \{(P_1, L, V)\} \quad \text{if } \rho, S \models b \qquad \text{IF1}$$

$$\rho, S, \mathscr{P} \uplus \{(\mathbf{if}\ b\ \mathbf{then}\ P_1\ \mathbf{else}\ P_2, L, V)\} \to \rho, S, \mathscr{P} \uplus \{(P_2, L, V)\} \quad \text{if } \rho, S \not\models b \qquad \text{IF2}$$

$$\rho, S, \mathscr{P} \uplus \{(\mathbf{set}(b^+); P, L, V)\} \to \rho, update(S, \rho(b^+)), \mathscr{P} \uplus \{(P, L, V)\} \qquad \text{SET}$$

$$\rho, S, \mathscr{P} \uplus \{(\mathbf{lock}(L'); P, L, V)\} \to \rho, S, \mathscr{P} \uplus \{(P, L \cup L', V)\}$$
$$\text{if } \forall\, (P'', L'', V'') \in \mathscr{P}\ .\ L' \cap L'' = \emptyset \qquad \text{LCK}$$

$$\rho, S, \mathscr{P} \uplus \{(\mathbf{unlock}(L'); P, L, V)\} \to \rho, S, \mathscr{P} \uplus \{(P, L \setminus L', V)\} \quad \text{if } L' \subseteq L \qquad \text{ULCK}$$

$\rho, S \models b_1 \wedge b_2$ iff $\rho, S \models b_1$ and $\rho, S \models b_2$
$\rho, S \models b_1 \vee b_2$ iff $\rho, S \models b_1$ or $\rho, S \models b_2$
$\rho, S \models \neg b$ iff $\rho, S \not\models b$
$\rho, S \models M \in s_i$ iff $\rho(M) \in S(s_i)$
$update(S, M \in s) = S \cup \{(s, M)\}$
$update(S, M \notin s) = S \setminus \{(s, M)\}$
$update(S, b_1^+; b_2^+) = update(update(S, b_1^+), b_2^+)$
$pt_x^V(a) = x_{a,V}$
$pt_x^V(f(T_1, \ldots, T_n)) = f(pt_x^{1::V}(T_1), \ldots, pt_x^{n::V}(T_n))$
$ri(V)$ denotes the set of variables x_k in V produced by replication.

Figure 6.4: Semantics for the process algebra

The rule PAR splits the process into two parallel processes. Replication REPL is annotated with $k \in \mathbb{N}$ and produces a fresh copy of P, adding x_k to V and the substitution $\{^k/_{x_k}\}$ to the environment ρ; the replication process is annotated with the index $k+1$ after the transition.

The rule for restriction NEW maps x to $a^l[V]$ in the continuation of the process, where l is a unique label for the process $\mathbf{new}^l\ x : a; P$. Here we extend the terms of Figure 6.1 to annotate names with a list of variables V under square brackets.

The rules for let reduce the process to P_1 where x is substituted with the result of the rewrite rule in case of success, and to P_2 otherwise.

To that end, we define the relation $\rightarrow_\rho$ as follows. Let s be a term that has only variables of atomic types and such that $\rho(s)$ is ground. Then $s \rightarrow_\rho t$ holds iff for some reduction rule $\mathbf{reduc}\ \forall \vec{x} : \vec{T}\ .\ l \rightarrow r$, there is a σ such that:

- σ is the most general unifier of l and s; w.l.o.g. we can assume that $fv(Img(\sigma)) \subseteq Dom(\rho)$;
- $t = \sigma(r)$. (Note that $\rho(t)$ is ground.)

Otherwise (if no such σ exists), we write $s \not\rightarrow_\rho$.

The rules for $\mathbf{if}\ b\ \mathbf{then}\ P_1\ \mathbf{else}\ P_2$ execute P_1 in case the set-membership state S satisfies the boolean formula b, and P_2 otherwise. The rule for **set** updates the current state according to the expression b^+. Finally, **lock** and **unlock** respectively acquire and release the locks on the sets in L' for the current process.

Having presented the semantic for Set-Pi, we need to prove that our typing judgements are preserved over the transition relation. Hence we introduce Lemma 1 to then prove subject reduction (Theorem 9).

We define the mapping $\widehat{a^l[V]} = a$ that recovers the type from an instrumented name, and its extension to environments $\widehat{\rho}(x) = \widehat{\rho(x)}$.

LEMMA 1 (TYPE SUBSTITUTION) *Let P be a process, Γ and Γ' two type environments, M a term and T a type. If $x \notin Dom(\Gamma')$ and $L, \Gamma[x \mapsto T]\Gamma' \vdash P$ and $\Gamma\Gamma' \vdash M : T$ then $L, \Gamma\Gamma' \vdash P\{^M/_x\}$.*

PROOF.[Proof sketch] The proof is carried out by induction on the shape of P, and its sub-terms and boolean conditions.

In particular, when x is encountered in P, we know that:

$$\frac{(\Gamma[x \mapsto T]\Gamma')(x) = T}{\Gamma[x \mapsto T]\Gamma' \vdash x : T}$$

is applied for the proof of $L, \Gamma[x \mapsto T]\Gamma' \vdash P$. Since $x\{^M/_x\} = M$, our statement directly follows from the hypothesis.

LEMMA 2 (ENVIRONMENT EXTENSION) *Let ρ and ρ' be two environments such that $Dom(\rho) \subseteq Dom(\rho')$ and for all x in $Dom(\rho)$ we have $\rho(x) = \rho'(x)$, let P be a process, Γ a type environment. If $L, \Gamma[\widehat{\rho}] \vdash P$, then $L, \Gamma[\widehat{\rho'}] \vdash P$.*

PROOF.By induction on the shape of P.

THEOREM 3 (SUBJECT REDUCTION) *Let Γ be a type environment, $\rho, S, \mathscr{P}$ a configuration. If for all $(P, L, V) \in \mathscr{P}$ we have $L, \Gamma[\widehat{\rho}] \vdash P$, and if $\rho, S, \mathscr{P} \to \rho', S', \mathscr{P}'$, then for all $(P', L', V') \in \mathscr{P}'$ we have $L', \Gamma[\widehat{\rho'}] \vdash P'$.*

PROOF.[Proof sketch] The statement can be proven by a case-by-case analysis of the semantic step $\rho, S, \mathscr{P} \to \rho', S', \mathscr{P}'$. A detailed version of this proof is available in the appendix.

Subject reduction enforces that well-typed processes remain well-typed over transitions, and in particular that values of the correct type are inserted and removed from sets, and that **if** and **set** only occur when the concerned sets are locked.

Attacker processes Because sets represent private information of the protocol, our attacker model does not use sets. Hence a valid attacker process is a well-typed process that shares a channel x_{ch} with the honest protocol, and cannot perform set operations (**if**, **set**, **lock** and **unlock** constructs are excluded).

6.5 Authentication in Set-Pi

A general definition of authentication goals for security protocols that has become standard in formal verification are Lowe's notions of non-injective and injective agreement [Low97]. These notions are hard to combine with an abstract interpretation approach as they inherently include a form of negation that is incompatible with over-approximation. For this reason, ProVerif has a special notion of *events* that are handled in a special way by its resolution procedure. We now show that in our Set-Pi calculus

we can directly express both non-injective and injective agreement using sets, and we can thus define *events* practically as syntactic sugar.

The definition of authentication is based on events $e_1(M)$ and $e_2(M)$ where message M typically contains the (claimed) sender and (intended) receiver name, as well as the data that the participants want to agree on; the event $e_1(M)$ is issued by the sender typically at the beginning of the session and $e_2(M)$ by the receiver at the end of the session, but the precise content and placement can be chosen by the modeler. One can then define non-injective agreement as follows: whenever an event $e_2(M)$ happens for a message M, then previously the event $e_1(M)$ must have happened. Thus, it is an attack if somebody accepts a message that has not been sent that way. The injective agreement additionally requires that if $e_2(M)$ has occurred n times, then $e_1(M)$ must have previously occurred at least n times (i.e., there is an injective mapping from e_2 events to e_1 events). Roughly speaking, it is an attack if a message is accepted more often than it was actually sent. It is hard to automatically verify injective agreement in this formulation. To simplify matters, it is common to require that the authenticated message M includes something fresh, i.e., a unique identifier that the sender chooses [The03, Bla09]. Thanks to this construction, the same e_1 event cannot occur more than once. Then, the injective agreement goal boils down to checking that no e_2 event occurs twice (and that non-injective agreement holds).

Let us thus extend Set-Pi with event declarations:

$Sys ::= \ldots \mid \textbf{new}\ e : \textbf{event}(T); Sys$

and event processes:

$P, Q ::= \ldots \mid \textbf{event}\ e(M); P$

and introduce the semantic rule (EVT):

$$\rho, S, \mathscr{P} \uplus \{(\textbf{event}\ e(M); P, L, V)\} \stackrel{e(M)}{\rightarrow} \rho, S, \mathscr{P} \uplus \{(P, L, V)\}$$

With this extension of the language, we can reason about non-injective and injective agreement properties according to Lowe's definitions. We then encode processes in the extended calculus with events into processes in standard Set-Pi and show how our encoding simulates the events.

DEFINITION 3 (NON-INJECTIVE AGREEMENT) *There is a non-injective agreement between* **event** $e_1(M)$ *and* **event** $e_2(M)$ *if and only if, for every possible trace* $\rho, S_0, \mathscr{P}_0 \rightarrow \rho, S_1, \mathscr{P}_1 \rightarrow \cdots \rightarrow \rho, S_n, \mathscr{P}_n$ *produced by the protocol, if* $\rho_i, S_i, \mathscr{P}_i \stackrel{e_2(M)}{\rightarrow} \rho_{i+1}, S_{i+1}, \mathscr{P}_{i+1}$ *occurs in the trace, then also* $\rho_j, S_j, \mathscr{P}_j \stackrel{e_1(M)}{\rightarrow} \rho_{j+1}, S_{j+1}, \mathscr{P}_{j+1}$ *occurs, for* $j < i$.

We construct a transformation from the extended language with events into the language without events, then prove the equivalence between Definition 3 in the original process (extended with events) and a set-property of the transformed process.

The transformation is as follows:

$$\begin{array}{lcl} \textbf{new } e : \textbf{event}(T); Sys & \rightarrow & \textbf{new } e : \textbf{set } T; Sys \\ \textbf{event } e(M); P & \rightarrow & \textbf{lock}(e); \textbf{set}(M \in e); \\ & & \textbf{unlock}(e); P \end{array}$$

Every event declaration becomes a set declaration in the translated process (assuming that the names for sets and events are disjoint). Whenever an **event** $e(M); P$ occurs, where M is of type T, we substitute it with the process that locks e, inserts M in the set e, unlocks e and continues with P; we also add a set declaration for e in its scope. Furthermore, to gain precision in the analysis we merge a set transition followed by an event into a single operation. That is, if we have a process $\textbf{set}(b^+); \textbf{event } e(M); P$, we transform it into the process $\textbf{lock}(e); \textbf{set}(b^+; M \in e); \textbf{unlock}(e); P$.

Note that this transformation is sound, although two semantic steps are merged into one: for the purpose of finding violations to an agreement property where $e(M)$ should happen before $e'(M)$, if there is a trace where **event** $e'(M)$ happens between $\textbf{set}(b^+)$ and **event** $e(M)$, then there is also a trace where **event** $e'(M)$ happens before the set operation. Given a process P we denote its event-free encoding as $agree(P)$.

THEOREM 4 *Let P be an extended process with events. If there is no reachable state S from $P' = agree(P)$ that satisfies the expression $M \in e_2 \wedge \neg M \in e_1$, then there is an non-injective agreement between $e_1(M)$ and $e_2(M)$ in P.*

PROOF.[Proof sketch] To prove the correctness of our transformation we construct a simulation relation between P and P', where the semantic step of an event is simulated by our construction. The full proof is found in the appendix.

DEFINITION 4 (INJECTIVE AGREEMENT) *There is an injective agreement between* **event** $e_1(M)$ *and* **event** $e_2(M)$ *if and only if, for every possible trace $\rho, S_0, \mathscr{P}_0 \rightarrow \rho, S_1, \mathscr{P}_1 \rightarrow \cdots \rightarrow \rho, S_n, \mathscr{P}_n$ produced by the protocol, if $\rho_i, S_i, \mathscr{P}_i \overset{e_2(M)}{\rightarrow} \rho_{i+1}, S_{i+1}, \mathscr{P}_{i+1}$ occurs in the trace, then also $\rho_j, S_j, \mathscr{P}_j \overset{e_1(M)}{\rightarrow} \rho_{j+1}, S_{j+1}, \mathscr{P}_{j+1}$ occurs, for some $j < i$; furthermore, there does not exists $k > i$ such that $\rho_k, S_k, \mathscr{P}_k \overset{e_2(M)}{\rightarrow} \rho_{k+1}, S_{k+1}, \mathscr{P}_{k+1}$.*

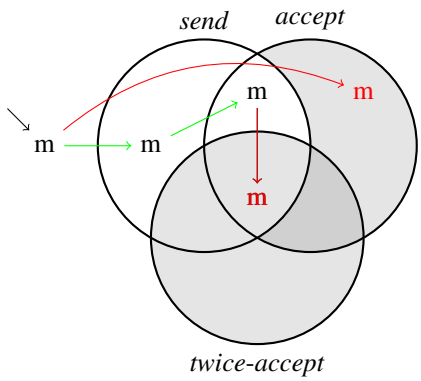


Figure 6.5: State transitions of messages in the CANAuth example

For proving injective agreement properties the transformation becomes:

$$
\begin{array}{lcl}
\textbf{new } e : \textbf{event}(T); \mathit{Sys} & \rightarrow & \textbf{new } e : \textbf{set } T; \\
 & & \textbf{new } \mathit{twice\text{-}e} : \textbf{set } T; \mathit{Sys} \\
\textbf{event } e(M); P & \rightarrow & \textbf{lock}(e, \mathit{twice\text{-}e}); \\
 & & \textbf{if } \neg M \in e \textbf{ then} \\
 & & \quad \textbf{set}(M \in e); \\
 & & \quad \textbf{unlock}(e, \mathit{twice\text{-}e}); P \\
 & & \textbf{else} \\
 & & \quad \textbf{set}(M \in \mathit{twice\text{-}e}); \\
 & & \quad \textbf{unlock}(e, \mathit{twice\text{-}e}); P
\end{array}
$$

Every event declaration becomes a pair of set declarations for e and *twice-e* in the translated process (assuming that the names for sets and events are disjoint). Whenever an **event** $e(M); P$ occurs, where M is of type T, we substitute it with the process that locks e and *twice-e*, and performs **set**$(M \in e); P$ when M has not yet been inserted in e; when it is already present in e it performs **set**$(M \in \mathit{twice\text{-}e}); P$, and in both cases unlocks e and *twice-e*; finally we add set declarations for e and *twice-e* in the scope. Similarly to the non-injective case, we merge a set operation with the event that follows. Given a process P, we denote its event-free encoding as $\mathit{inj\text{-}agree}(P)$.

THEOREM 5 *Let P be an extended process with events. If no reachable state S from $P' = \mathit{inj\text{-}agree}(P)$ satisfies the expression $(M \in e_2 \wedge \neg M \in e_1) \vee (M \in \mathit{twice\text{-}e}_2)$, then the injective agreement between $e_1(M)$ and $e_2(M)$ holds in P.*

PROOF.[Proof sketch] To prove the correctness of our transformation we construct a simulation relation between P and P', where the semantic step of an event is simulated by our construction. The full proof is found in the appendix.

Relating back to our example, Figure 6.5 shows in green the desired transitions and in red the undesired ones. Our model satisfies non-injective agreement if no message is being accepted without being previously sent by the honest principal. It satisfies injective agreement if no message is accepted twice.

6.6 Translation

The translation takes a process in Set-Pi and produces a set of Horn clauses that are then solved by a saturation based resolution engine, like ProVerif or SPASS.

At the end of the section we show how the translation is carried out for our CANAuth example. We now present the general concepts of the translation at an intuitive level, which we then refine with details later in the section. The translation produces clauses with predicates of the form $\mathsf{msg}(M,N)$ to denote that the system has produced an output of N on channel M, predicates of the form $\mathsf{att}(M)$ to denote that the attacker process knows M, predicates of the form $\mathsf{name}(a)$ to denote that a new name is produced by the protocol, and clauses that conclude $\mathsf{implies}(\cdot,\cdot)$ to denote set-transitions.

The body of a Horn clause represents the inputs that are required to reach a specific point in the process, while the head of the clause represents the output that is generated. For example:

$$\mathbf{in}(ch,x:a);\mathbf{in}(ch,y:b);\mathbf{out}(ch,f(x,y))$$

produces the clause:

$$\mathsf{msg}(ch,x)\wedge\mathsf{msg}(ch,y)\Rightarrow\mathsf{msg}(f(x,y))$$

Names and variables in the predicates are annotated with a special constructor *val* that defines their current membership class. For example, if we have three sets in our system $\mathsf{s}_1,\mathsf{s}_2,\mathsf{s}_3$, the term $val(a,1,0,x_{\mathsf{s}_3,a})$ represents a name a in the process algebra in a state where a is in s_1, it is not in s_2 and its membership to the set s_3 is not constrained, as denoted by the variable $x_{\mathsf{s}_3,a}$. By doing so two clauses can be unified only if the terms are in consistent states. For example, $val(a,1,0,x_{\mathsf{s}_3,a})$ unifies with $val(a,x_{\mathsf{s}_1,a},0,1)$ but not with $val(a,x_{\mathsf{s}_1,a},1,1)$, because the first term represents a name a that is not in s_2, while the third term represents a in a state where it belongs to s_2.

Now we look at how the translation is constructed. We use a special function α, which we call the set-abstraction, to record whether a particular term belongs or not to some sets, and introduce the rules of Figure 6.6 to transform clauses, predicates and terms into annotated ones. The set-abstraction is a function of type:

$$\alpha:(\mathbb{S}\times\mathbb{M})\rightarrow(\{0,1\}\cup\{x_{s,M}\mid s\in S,M\in M\})$$

$$\langle\!|H_1 \wedge \cdots \wedge H_n \Rightarrow C|\!\rangle_\alpha = \langle\!|H_1|\!\rangle_\alpha \wedge \cdots \wedge \langle\!|H_n|\!\rangle_\alpha \Rightarrow \langle\!|C|\!\rangle_\alpha$$
$$\langle\!|\mathsf{p}(M_1,\ldots,M_n)|\!\rangle_\alpha = \mathsf{p}(\langle\!|M_1|\!\rangle_\alpha,\ldots,\langle\!|M_n|\!\rangle_\alpha)$$
$$\langle\!|f(M_1,\ldots,M_n)|\!\rangle_\alpha = f(\langle\!|M_1|\!\rangle_\alpha,\ldots,\langle\!|M_n|\!\rangle_\alpha)$$
$$\langle\!|a^l[V]|\!\rangle_\alpha = \begin{cases} val(a^l[V],\alpha(\mathsf{s}_1,a^l[V]),\ldots,\alpha(\mathsf{s}_n,a^l[V])) & \text{if } l \in labels(P_0) \\ val(a^\top[],\alpha(\mathsf{s}_1,a^l[V]),\ldots,\alpha(\mathsf{s}_n,a^l[V])) & \text{otherwise} \end{cases}$$
$$\langle\!|x|\!\rangle_\alpha = val(x,\alpha(\mathsf{s}_1,x),\ldots,\alpha(\mathsf{s}_n,x))$$

Figure 6.6: Applying the set-abstraction

where we require:

$$\alpha(s,M) \notin \{0,1\} \implies \alpha(s,M) = x_{s,M}$$

It takes a process set s and a term M, and returns either the constant 1, to enforce that M is in s, the constant 0 , to enforce that M is not in s, or the variable $x_{s,M}$, to allow one of the two choices to be picked consistently across the hypotheses.

The function $\langle\!|p|\!\rangle_\alpha$ of Figure 6.6 recursively applies the set-abstraction to the clauses. When it encounters an annotated name $a^l[V]$ in the protocol, it produces a constructor $val(\ldots)$ where the first parameter is the name itself — with no annotations in case of attacker names— and the remaining parameters represent the membership of $a^l[V]$ to the sets $s_1,\ldots,s_n$; similar clauses are generated for occurring variables. For the purpose of making the analysis feasible, as a well-formedness condition we require set types (of the form **set** T) to contain only name types and monadic constructors over name types. For example, **set** *Seed*, **set** $pk(Seed)$ and **set** $sk(Seed)$ are acceptable set types, while **set** $key(Seed,Nonce)$ is not.

The function $[\![P]\!]HVL\alpha$ of Figure 6.8 takes a process P, a set of hypothesis predicates H, that intuitively represent the set of messages required to reach P, a list of influencing terms for the process V, a set of locks L held by the process, and the set-abstraction α, and produces a set of clauses representing the protocol behaviour.

Lastly the functions *restrict*, *zero* and *relax* of Figure 6.6 modify the set-abstraction for various constructs of Set-Pi. The function *restrict* takes a set-abstraction α and a boolean formula b and produces the set of all consistent abstractions that satisfy b, while *zero* inserts the constant 0 for fresh names, and *relax* introduces variables in the image of α for unlocked sets.

Having introduced the auxiliary functions for manipulating the set-abstraction, we now come back to explaining the translation process. The function $relax(\alpha,L)$ is applied at

$$restrict(\alpha, M \in s) = \text{if } \alpha(s,M) \neq 0 \text{ then } \{\alpha'\} \text{ else } \emptyset$$
$$\text{where } \alpha'(s',M') = \begin{cases} 1 & \text{if } M' = M \wedge s' = s \\ \alpha(s',M') & \text{otherwise} \end{cases}$$
$$restrict(\alpha, \neg M \in s) = \text{if } \alpha(s,M) \neq 1 \text{ then } \{\alpha'\} \text{ else } \emptyset$$
$$\text{where } \alpha'(s',M') = \begin{cases} 0 & \text{if } M' = M \wedge s' = s \\ \alpha(s',M') & \text{otherwise} \end{cases}$$
$$restrict(\alpha, b_1 \wedge b_2) = \bigcup_{\alpha' \in restrict(\alpha,b_1)} restrict(\alpha', b_2)$$
$$restrict(\alpha, b_1 \vee b_2) = restrict(\alpha, b_1) \cup restrict(\alpha, b_2)$$
$$restrict(\alpha, \neg(b_1 \wedge b_2)) = restrict(\alpha, (\neg b_1) \vee (\neg b_2))$$
$$restrict(\alpha, \neg(b_1 \vee b_2)) = restrict(\alpha, (\neg b_1) \wedge (\neg b_2))$$
$$restrict(\alpha, \neg\neg b) = restrict(\alpha, b)$$
$$zero(\alpha, a) = \alpha'$$
$$\text{where } \alpha'(s,M) = \begin{cases} 0 & \text{if } a \text{ occurs in } M \\ \alpha(s,M) & \text{otherwise} \end{cases}$$
$$relax(\alpha, L) = \alpha'$$
$$\text{where } \alpha'(s,M) = \begin{cases} \alpha(s,M) & \text{if } s \in L \\ x_{s,M} & \text{otherwise} \end{cases}$$

Figure 6.7: Functions for updating α

each step of the translation, as it inserts variables in the image of α for all sets that are not locked, as they may be changed by other processes.

The translation for 0 produces an empty set of clauses. Replication $!^l P$ translates P with the introduction of a new session variable x_l in the list of influencing variables V. Parallel composition $P_1 \,|\, P_2$ is translated as the union of the clauses generated by both processes.

Input $\mathbf{in}(M, x:T)$ adds the predicate $\mathsf{msg}(M,N')$ as an hypothesis in H, where N' is a copy of T where every occurrence of a name type is replaced with a unique variable using $pt_x^{ri(V)}(T)$; the substitution $\{^{N'}/_x\}$ is then applied on the continuation. Output $\mathbf{out}(M,N)$ produces a clause that generates $(\!|\mathsf{msg}(M,N)|\!)_\alpha$ if all hypotheses in $(\!|H|\!)_\alpha$ are satisfied. The rule for $\mathbf{new}^l\ x:a$ introduces a restricted name: the value class of $a^l[V]$ is set to 0 for every set, the predicate $\mathsf{name}(a^l[V])$ is introduced both in the hypotheses

$[\![0]\!]HVL\alpha = \emptyset$

$[\![!^l P]\!]HV\emptyset\alpha = [\![P]\!]H(x_l :: V)\emptyset(relax(\alpha,\emptyset))$

$[\![P_1 \mid P_2]\!]HV\emptyset\alpha = [\![P_1]\!]HV\emptyset(relax(\alpha,\emptyset)) \cup [\![P_2]\!]HV\emptyset(relax(\alpha,\emptyset))$

$[\![\mathbf{in}(M,x:T);P]\!]HVL\alpha = [\![P\{^{N'}/_x\}]\!](H \wedge \mathsf{msg}(M,N'))(N' :: V)L\alpha'$
where $\alpha' = relax(\alpha,L)$, $N' = pt_x^{ri(V)}(T)$

$[\![\mathbf{out}(M,N);P]\!]HVL\alpha = [\![P]\!]HVL(relax(\alpha,L)) \cup \{\langle\!|H \Rightarrow \mathsf{msg}(M,N)|\!\rangle_\alpha\}$

$[\![\mathbf{new}^l\ x:a;^l P]\!]HVL\alpha = [\![P\{^{a^l[V]}/_x\}]\!](H \wedge \mathsf{name}(a^l[V]))VL\alpha' \cup \{\langle\!|H \Rightarrow \mathsf{name}(a^l[V])|\!\rangle_{\alpha'}\}$
where $\alpha' = zero(\alpha, a^l[V])$

$[\![\mathbf{let}\ x = g(M_1,\ldots,M_n)\ \mathbf{in}\ P_1\ \mathbf{else}\ P_2]\!]HVL\alpha =$
$\{[\![\sigma(P_1)]\!]\sigma(H)\sigma(V)L\alpha'' \mid \mathbf{reduc}\ \forall \vec{x'} : \vec{T'}\ .\ g(M'_1,\ldots,M'_n) \to M';$ is in the scope of **let**,
σ is an m.g.u. that satisfies $M_1 \stackrel{\circ}{=} M'_1 \wedge \cdots \wedge M_n \stackrel{\circ}{=} M'_n \wedge x \stackrel{\circ}{=} M'$,
θ is an m.g.u. that satisfies $\forall s,N_1,N_2,\ \sigma(N_1) = \sigma(N_2) \Rightarrow \alpha'(s,N_1) \stackrel{\circ}{=} \alpha'(s,N_2)$,
and α'' satisfies $\forall N\,.\,\alpha''(s,\sigma(N)) = \theta(\alpha'(s,N))\} \cup [\![P_2]\!]HVL\alpha'$
where $\alpha' = relax(\alpha,L)$

$$[\![\mathbf{if}\ b\ \mathbf{then}\ P_1\ \mathbf{else}\ P_2]\!]HVL\alpha = \bigcup_{\alpha' \in restrict(relax(\alpha,L),b)} [\![P_1]\!]HVL\alpha' \quad \cup \quad \bigcup_{\alpha' \in restrict(relax(\alpha,L),\neg b)} [\![P_2]\!]HVL\alpha'$$

$[\![\mathbf{lock}(L');P]\!]HVL\alpha = [\![P]\!]HV(L \cup L')(relax(\alpha,L))$

$[\![\mathbf{unlock}(L');P]\!]HVL\alpha = [\![P]\!]HV(L \setminus L')(relax(\alpha,L))$

$[\![\mathbf{set}(b^+);P]\!]HVL\alpha = \{\langle\!|H|\!\rangle_{\alpha'} \Rightarrow \mathsf{implies}(\langle\!|M|\!\rangle_{\alpha'}, \langle\!|M|\!\rangle_{\alpha''}) \mid M \in fv(b^+) \cup fn(b^+)\} \cup [\![P]\!]HVL\alpha'''$
where $\alpha' = relax(\alpha,L)$ and $\alpha'' = update(\alpha,b^+)$ and $\alpha''' = relax(\alpha'',L)$

Figure 6.8: Translation of Set-Pi into Horn clauses

for analysing the continuation and as a fact that follows the current set of hypotheses H. This ensures that all the set-abstraction variables occurring in the head of a clause are closed under the hypotheses.

The rule for **let** $x = g(M_1,\ldots,M_n)$ **in** P_1 **else** P_2 looks for a substitution σ that successfully unifies a definition of the rewrite rule for the destructor g with the actual parameters $M_1,\ldots,M_n$, and then finds a substitution θ that unifies the terms in the set-abstraction α accordingly to the unification on the process algebra terms; if both substitutions are found then $\sigma(P_1)$ is analysed where x is substituted with the result of the reduction.

The rule for **if** b **then** P_1 **else** P_2 translates P_1 with all the set-abstractions that satisfy the formula b, and P_2 with all the set-abstractions that satisfy the formula $\neg b$. The rule for **lock**(s); translates the continuation by first introducing s in the locked sets

L, and applying *relax* to take into account state changes from other processes before the lock takes place. Similarly, the rule for $\mathbf{unlock}(s)$; translates the continuation by removing s from the set L, and applying *relax*. The rule for $\mathbf{set}(b^+)$; P, for every name and variable occurring in b^+ that we denote by M, creates a clause of the form $\langle\!\langle H \rangle\!\rangle_{\alpha'} \Rightarrow \mathsf{implies}(\langle\!\langle M \rangle\!\rangle_{\alpha'}, \langle\!\langle M \rangle\!\rangle_{\alpha' \cup \{(s,M)\}})$, and translates the continuation. The clauses produced by this last rule ensure that whenever M appears in a predicate on state α', we will also have the same predicate on state $\alpha' \cup \{(s,M)\}$.

Clauses representing the attacker We add the following set of clauses to represent a Dolev-Yao attacker. The attacker can eavesdrop messages form known channels:

$$\mathsf{msg}(x_{ch}, x_{msg}) \wedge \mathsf{att}(x_{ch}) \Rightarrow \mathsf{att}(x_{msg})$$

The attacker can insert known messages into channels:

$$\mathsf{att}(x_{ch}) \wedge \mathsf{att}(x_{msg}) \Rightarrow \mathsf{msg}(x_{ch}, x_{msg})$$

For every n-ary constructor f occurring in the protocol we produce a clause:

$$\mathsf{att}(x_1) \wedge \cdots \wedge \mathsf{att}(x_n) \Rightarrow \mathsf{att}(f(x_1, \ldots, x_n))$$

For all destructors of the form $g(M_1, \ldots, M_n) \rightarrow M$ we produce a clause:

$$\mathsf{att}(M_1) \wedge \cdots \wedge \mathsf{att}(M_n) \Rightarrow \mathsf{att}(M)$$

Finally, the attacker knows a name for each name type a in the initial state S_0:

$$\Rightarrow \langle\!\langle \mathsf{att}(a^\top[]) \rangle\!\rangle_{S_0}; \quad \Rightarrow \langle\!\langle \mathsf{name}(a^\top[]) \rangle\!\rangle_{S_0}$$

as well as the public channel shared with the honest protocol:

$$\Rightarrow \langle\!\langle \mathsf{att}(ch^{l_0}[]) \rangle\!\rangle_{S_0}$$

Clauses representing the set-transitions Let Cl be the set of clauses produced by the translation $[\![P_0]\!]\emptyset\emptyset\emptyset\alpha_0$. In order to transfer the knowledge obtained between states, for every predicate $\mathsf{p}(C[a])$ used as conclusion in Cl, for every compatible conclusion of the form $\mathsf{implies}(\langle\!\langle a \rangle\!\rangle_\alpha, \langle\!\langle a \rangle\!\rangle_{\alpha'})$ in Cl, we produce a set of clauses of the form:

$$\langle\!\langle \mathsf{p}(C[a]) \rangle\!\rangle_\alpha \wedge \mathsf{implies}(\langle\!\langle a \rangle\!\rangle_\alpha, \langle\!\langle a \rangle\!\rangle_{\alpha'}) \Rightarrow \langle\!\langle \mathsf{p}(C[a]) \rangle\!\rangle_{\alpha'}$$

This set of clauses transfers the attacker knowledge between state transitions. Intuitively this set of rules suffices for the translation because only the honest protocol produces state transitions, and everything that the attacker can derive in a state, it can also derive in the successor state. Therefore, it is only necessary to transfer the hypotheses for the protocol. Lemma 9 in Appendix A.3 establishes the correctness of this approach.

Translation of CANAuth into Horn clauses To show how the translation is applied to produce Horn clauses from the original description, we have taken an excerpt from our running example, namely the receiving process, translated the events into set transitions, and labelled each point of the program:

$$
\begin{aligned}
P_1 \triangleq\ & !^{l_1\ \{r,a,at\}\ l_2}\mathbf{in}(ch, \langle x_m, x_s\rangle : \\
& \quad \langle msg(cnt), hmac(key, msg(cnt))\rangle);^{l_3} \\
& \mathbf{let}\ \_ = eq(x_s, hmac(k, x_m))\ \mathbf{in}^{l_4} \\
& \mathbf{let}\ x_c = getcnt(x_m)\ \mathbf{in}^{l_5} \\
& \quad \mathbf{if}\ x_c \notin r\ \mathbf{then}^{l_6} \\
& \quad\quad \mathbf{if}\ x_m \notin a\ \mathbf{then}^{l_7}\mathbf{set}(x_c \in r; x_m \in a);^{l_8} \\
& \quad\quad \mathbf{else}^{l_9}\mathbf{set}(x_c \in r; x_m \in at);^{l_{10}}
\end{aligned}
$$

Figure 6.9 shows the clauses that are generated, together with the recursive calls of $[\![\cdot]\!]$ that are required to produce them. We use here the notation like $(H_1 = \emptyset)$ to indicate the development of the parameters H, V, L and α over the recursive calls of $[\![\cdot]\!]$, and P^l to denote the subprocess of P_1 at label l.

6.7 Correctness

In this section we want to establish the correctness of our translation with respect to the semantics of Section 6.4. We use the inference system of Figure 6.10 to express our correctness results. Intuitively, this set of rules relates the instrumented semantics to the Horn clauses generated by the translation, namely that the fixed-point $\mathscr{F}_{P_0}$ covers all possible behaviours of a process, when started in the given configuration (ρ, L, V, S) and in any environment that cannot change sets in L.

Let S and S' again be states of the sets (i.e., $S(s)$ yields the elements that are members of set s in state S); we can view a state as a special case of an abstraction α that has no variables (i.e., indetermined set memberships) and we can thus write $\langle\!|\cdot|\!\rangle_S$ accordingly. We will now show: if the semantic relation induces a reachable state S at which output N on channel M is produced, then the Horn clauses generated for the protocol entails the ground fact $\langle\!|\mathsf{msg}(M,N)|\!\rangle_S$. This ensures that whatever behaviour is present in the semantics is also captured by the translation.

We denote by $\mathscr{C}_{P_0}$ the set of clauses produced by the translation, including the fixed clauses, and by $\mathscr{F}_{P_0}$ the set of ground facts derivable from $\mathscr{C}_{P_0}$. First we introduce the order relation $\preceq_L$ on the facts $\mathscr{F}_{P_0}$ derivable from the initial protocol.

$$
\begin{aligned}
[\![P_1]\!]\emptyset[]\emptyset\alpha_0 &= [\![P^{l_1}]\!](H_1 = \emptyset)(V_1 = [x_{l_1}])(L_1 = \emptyset)(\alpha_1 = relax(\alpha_0, \emptyset)) \\
&= [\![P^{l_2}]\!](H_2 = \emptyset)(V_2 = V_1)(L_2 = \{r, a, at\})(\alpha_2 = relax(\alpha_1, \emptyset)) \\
&= [\![P^{l_3}]\!](H_3 = \{\mathsf{msg}(ch[], T_3 = \langle msg(x_{cnt,1}), hmac(x_{key,3}, msg(x_{cnt,2}))\rangle)\}) \\
&\qquad (V_3 = T_3 :: V_2)(L_3 = L_2)(\alpha_3 = relax(\alpha_2, L_2)) \\
&= [\![P^{l_4}]\!](H_4 = \{\mathsf{msg}(ch[], T_4 = \langle msg(x_{cnt,1}), hmac(K[], msg(x_{cnt,1}))\rangle)\}) \\
&\qquad (V_4 = T_4 :: V_2)(L_4 = L_3)(\alpha_4 = \theta(relax(\alpha_3, L_3))) \\
&= [\![P^{l_5}]\!](H_5 = H_4)(V_5 = V_4)(L_5 = L_4)(\alpha_5 = relax(\alpha_4, L_4)) \\
&= [\![P^{l_6}]\!](H_6 = H_5)(V_6 = V_5)(L_6 = L_5)(\alpha_6 = restrict(relax(\alpha_5, L_5), x_{cnt,1} \notin r)) \\
&= [\![P^{l_7}]\!](H_7 = H_6)(V_7 = V_6)(L_7 = L_6)(\alpha_7 = restrict(relax(\alpha_6, L_6), msg(x_{cnt,1}) \notin a)) \cup \\
&\quad [\![P^{l_{10}}]\!](H_{10} = H_6)(V_{10} = V_6)(L_{10} = L_6)(\alpha_{10} = restrict(relax(\alpha_6, L_6), msg(x_{cnt,1}) \in a)) \\
[\![P^{l_7}]\!]H_7V_7L_7\alpha_7 &= \{\mathsf{msg}(msg(val(ch[])), \langle msg(val(x_1, x_{1,s}, 0, 0, x_{1,at})), hmac(val(k[]), msg(val(x_1, x_{1,s}, 0, 0, x_{1,at})))\rangle) \\
&\qquad \Rightarrow \mathsf{implies}(val(x_1, x_{1,s}, 0, 0, x_{1,at}), val(x_1, x_{1,s}, 1, 1, x_{1,at}))\} \\
[\![P^{l_{10}}]\!]H_{10}V_{10}L_{10}\alpha_{10} &= \{\mathsf{msg}(msg(val(ch[])), \langle msg(val(x_1, x_{1,s}, 0, 1, x_{1,at})), hmac(val(k[]), msg(val(x_1, x_{1,s}, 0, 1, x_{1,at})))\rangle) \\
&\qquad \Rightarrow \mathsf{implies}(val(x_1, x_{1,s}, 0, 1, x_{1,at}), val(x_1, x_{1,s}, 1, 1, 1))\}
\end{aligned}
$$

Figure 6.9: Horn clauses generated by the translation

$$\frac{}{\rho,V,L,S \Vdash 0}\ \text{T-NIL}$$

$$\frac{\forall S' \text{ s.t. } S \preceq_\emptyset S'\ (\rho,V,\emptyset,S' \Vdash Q_1 \wedge \rho,V,\emptyset,S' \Vdash Q_2)}{\rho,V,\emptyset,S \Vdash Q_1 \mid Q_2}\ \text{T-PAR}$$

$$\frac{\forall S' \text{ s.t. } S \preceq_\emptyset S'\ (\rho\{^l/_{x_l}\},(x_l :: V),\emptyset,S' \Vdash Q)}{\rho,V,\emptyset,S \Vdash !^l\, Q}\ l \in \mathbb{N}\ \text{T-REPL}$$

$$\frac{\begin{array}{l}\forall S' \text{ s.t. } S \preceq_L S'\ \forall N \text{ s.t. } \Gamma \vdash N : T \\ \{\!|\rho(\mathsf{msg}(M,N))|\!\}_{S'} \in \mathscr{F}_{P_0} \Rightarrow (mgu(N',N) \circ \rho), \\ (N' :: V),L,S' \Vdash Q\{^{N'}/_x\}\end{array}}{\rho,V,L,S \Vdash \mathbf{in}(M,x:T);Q}\ N' = pt_x^{ri(V)}(T)\ \text{T-IN}$$

$$\frac{\{\!|\rho(\mathsf{msg}(M,N))|\!\}_S \in \mathscr{F}_{P_0} \wedge \forall S' \text{ s.t. } S \preceq_L S'\ (\rho,V,L,S' \Vdash Q)}{\rho,V,L,S \Vdash \mathbf{out}(M,N);Q}\ \text{T-OUT}$$

$$\frac{\{\!|\rho(\mathsf{name}(a^l[V]))|\!\}_S \in \mathscr{F}_{P_0} \wedge \forall S' \text{ s.t. } S \preceq_L S'\ \ \rho,V,L,S' \Vdash Q\{^{a^l[V]}/_x\}}{\rho,V,L,S \Vdash \mathbf{new}^l\ x : a;Q}\ \text{T-NEW}$$

$$\frac{\begin{array}{l}\forall S' \text{ s.t. } S \preceq_L S'\ (\forall M \text{ s.t. } g(M_1,\ldots,M_n) \to_\rho M \\ \rho,V,L,S' \Vdash Q_1\{^M/_x\}) \wedge \rho,V,L,S' \Vdash Q_2\end{array}}{\rho,V,L,S \Vdash \mathbf{let}\ x = g(M_1,\ldots,M_n)\ \mathbf{in}\, Q_1\ \mathbf{else}\ Q_2}\ \text{T-LET}$$

$$\frac{\forall S' \text{ s.t. } S \preceq_L S',\ (\rho,S' \models b \Rightarrow \rho,V,L,S' \Vdash Q_1) \wedge (\rho,S' \models \neg b \Rightarrow \rho,V,L,S' \Vdash Q_2)}{\rho,V,L,S \Vdash \mathbf{if}\ b\ \mathbf{then}\, Q_1 \mathbf{else}\ Q_2}\ \text{T-IF}$$

$$\frac{\forall S' \text{ s.t. } S \preceq_L S'\ \rho,V,(L \cup L'),S' \Vdash Q}{\rho,V,L,S \Vdash \mathbf{lock}(L');Q}\ \text{T-LOCK}$$

$$\frac{\forall S' \text{ s.t. } S \preceq_L S'\ \rho,V,(L \setminus L'),S' \Vdash Q}{\rho,V,L,S \Vdash \mathbf{unlock}(L');Q}\ \text{T-UNLOCK}$$

$$\frac{\begin{array}{l}\forall S' \text{ s.t. } S \preceq_L S', (\forall\, M \in fv(b^+) \cup fn(b^+) \\ \mathsf{implies}(\{\!|\rho(M)|\!\}_{S'}, \{\!|\rho(M)|\!\}_{S''}) \in \mathscr{F}_{P_0}) \wedge \\ (\forall S''' \text{ s.t. } S'' \preceq_L S''',\ \rho,V,L,S''' \Vdash Q)\end{array}}{\rho,V,L,S \Vdash \mathbf{set}(b^+);Q}\ S'' = update(S',\rho(b^+))\ \text{T-SET}$$

Figure 6.10: Inference system for correctness

DEFINITION 5 (ORDER RELATION $\preceq_L$) *The order relation $S_1 \preceq_L S_2$ between states S_1 and S_2 holds iff:*

(i) $\forall s_j \in L \,.\, S_1(s_j) = S_2(s_j)$;

(ii) $\forall \mathsf{p}(M_1,\ldots,M_k) \,.\, (\!|\mathsf{p}(M_1,\ldots,M_k)|\!)_{S_1} \in \mathscr{F}_{P_0} \Rightarrow (\!|\mathsf{p}(M_1,\ldots,M_k)|\!)_{S_2} \in \mathscr{F}_{P_0}$.

Intuitively the $\preceq_L$ relation captures the causal relation of the semantic rules, as condition *(ii)* requires all predicates of the form msg, name and att to be transferred from state S_1 to state S_2. Furthermore condition *(i)* imposes that the locked sets L are not modified between the two states. The most general of such relations is $\preceq_\emptyset$, as it allows any set to be modified.

Next we formalise the definition for set-abstraction α that was introduced in Section 6.6.

DEFINITION 6 (SET-ABSTRACTION) *The mapping α abstracts S under the environment ρ iff for every set s, term M, either $\alpha(s,M) = 1$ and $\rho(M) \in S(s)$, or $\alpha(s,M) = 0$ and $\rho(M) \notin S(s)$, or $\alpha(s,M) = x_{s,M}$.*

A set abstraction α abstracts a state S if every pair (s,M) that maps to a variable in α is mapped to a variable that is unique in the image (this is ensured syntactically by the use of $x_{s,M}$), and whenever $\alpha(s,M)$ maps to the constants 1 and 0 then $\rho(M) \in s$ and $\rho(M) \notin s$, respectively, in the state S.

The following lemmata establish the relation between the operations used in the translation and the order relation $\preceq_L$. Formal proofs of these properties are present in the appendix.

LEMMA 3 (*relax* PRESERVES THE SET-ABSTRACTION OVER $\preceq_L$) *Let S, S' be two states such that $S \preceq_L S'$, and assume α abstracts S under ρ. Then $\alpha' = relax(\alpha, L)$ abstracts S' under ρ.*

Since *relax* inserts unique variables in α' for all sets that are not locked, and for all sets s that are locked $\alpha(s) = \alpha'(s)$ and $S(s) = S'(s)$ holds by condition *(i)* of the order relation, then α satisfies the properties of Definition 6.

LEMMA 4 (*restrict* PRESERVES THE SET-ABSTRACTION) *Let α be a set abstraction, ρ an environment, S a state and $A = restrict(\alpha, b)$. If $\rho, S \models b$ and α abstracts S, then there exists an $\alpha' \in A$ such that α' abstracts S.*

Restrict produces a set of set-abstractions each representing a possible way of satisfying the formula b. Lemma 8 establishes that if α abstracts S then at least one of these restrictions on α satisfies the abstraction of S.

LEMMA 5 (implies **PRESERVES** $S \preceq_L S'$) *Let S be a set-membership state, and $S' = S \cup \{(s_1, M_1), \ldots, (s_j, M_j)\} \setminus \{(s_{j+1}, M_{j+1}), \ldots, (s_n, M_n)\}$. If for all $M \in \{M_1, \ldots, M_n\}$ we have* $\mathsf{implies}(\langle\!|M|\!\rangle_S, \langle\!|M|\!\rangle_{S'}) \in \mathscr{F}_{P_0}$ *then for any set of locks L such that $\{s_1, \ldots, s_n\} \cap L = \emptyset$ we have $S \preceq_L S'$.*

Lemma 9 in Appendix A.3 establishes that implies predicates actually capture the state transitions, hence following the definition of the order $\preceq_L$ the set of predicates derivable in the updated state is larger than that derivable in the original state.

Next we type the attacker process A and the honest protocol P_0, under the initial environment $\rho_0 = [x_{ch} \mapsto ch^{l_0}[]]$.

LEMMA 6 (TYPABILITY OF A) *Let A be an attacker process, then $\rho_0, \emptyset, \emptyset, S_0 \Vdash A$.*

PROOF.[Proof of sketch] Let B be a subprocess of A, ρ an environment, S a state, V a list of terms. We prove that if:

(i) $\rho(B)$ is a closed process, $\rho(V)$ is ground,

(ii) $S_0 \preceq_\emptyset S$, and

(iii) for every maximal subterm M of B closed under ρ, we have $\langle\!|\rho(\mathsf{att}(M))|\!\rangle_S \in \mathscr{F}_{P_0}$,

then:

$$\rho, V, \emptyset, S \Vdash B$$

Proof by induction over the depth of B.

In particular, we have that (i) $fv(A) = x_{ch}$, hence $\rho_0(A)$ is closed; (ii) $S_0 \preceq_\emptyset S_0$ by reflexivity; and (iii) the only maximal subterm of A that is bound by ρ_0 is x_{ch}, and by construction of the translation $\langle\!|\rho_0(\mathsf{att}(x_{ch}))|\!\rangle_{S_0} \in \mathscr{F}_{P_0}$. Hence the attacker process types.

For the full proof we refer to the appendix.

LEMMA 7 (TYPABILITY OF P_0) $\rho_0, \emptyset, \emptyset, S_0 \Vdash P_0$.

PROOF.[Proof sketch] Let Q be a process. We prove that, given a list of terms V, a set of locks L, a state S, a set-abstraction α, an environment ρ; if:

(i) $\rho(Q)$ is a closed process, $\rho(V)$ and $\rho(H)$ are ground,

(ii) α abstracts S under ρ,

(iii) $\mathscr{C}_{P_0} \supseteq [\![Q]\!]HVL\alpha$,

(iv) for every predicate p in H, we have that $\{\!|\rho(p)|\!\}_S \in \mathscr{F}_{P_0}$

Then $\rho, V, L, S \Vdash Q$.

The proof is carried out by induction on the structure of the process Q.

In particular, (i) ρ_0 closes P_0 by construction, $\rho_0(\emptyset)$ is trivially ground, (ii) α_0 abstracts S_0 under ρ_0 by construction, (iii) $\mathscr{C}_{P_0} \supseteq [\![P_0]\!]\emptyset\emptyset\emptyset\alpha_0$ by definition of the translation, (iv) holds vacuously. Therefore the conditions (i–iv) are satisfied and hence $\rho_0, \emptyset, \emptyset, S_0 \Vdash P_0$.

For the full proof we refer to the appendix.

THEOREM 6 (SUBJECT REDUCTION) *If $\rho, S, \mathscr{P} \to \rho', S', \mathscr{P}'$ and for all $(P, L, V) \in \mathscr{P}$ we have $\rho, V, L, S \Vdash P$ then for all $(P', L', V') \in \mathscr{P}'$ we have $\rho', V', L', S' \Vdash P'$.*

The proof, found in the appendix, is a case-by-case analysis on the semantic rules for the language.

THEOREM 7 (CORRECTNESS OF THE ANALYSIS) *Let $Sys[\cdot]$ be the system context, let P_0 be the protocol, let T be the set of types used by P_0, let A be any attacker process using only types in T, and $\rho_0 = [x_{ch} \mapsto ch^{l_0}[]]$.*

If the typing $[] \vdash Sys[\mathbf{new}^{l_0}\, x_{ch} : ch; P_0 \,|\, A]$ holds, and if $\rho_0, S_0, \mathscr{P}_0 = \{(P_0 \,|\, A, \emptyset, \emptyset)\} \to^* \rho_n, S_n, \mathscr{P}_n = P_{n'} \uplus \{(\mathbf{out}(M, N); P', L, V)\}$; then $\{\!|\rho(\mathsf{msg}(M, N))|\!\}_{S_n} \in \mathscr{F}_{P_0}$.

PROOF.By Lemma 11 we know that $\rho_0, \emptyset, \emptyset, S_0 \Vdash P_0$; by Lemma 10 we know that $\rho_0, \emptyset, \emptyset, S_0 \Vdash A$, hence all processes in $\mathscr{P}_0$ type in the initial state S_0.

Let $\rho_0, S_0, \mathscr{P}_0 \to \rho_1, S_1, \mathscr{P}_1 \to \cdots \to \rho_n, S_n, \mathscr{P}_n$. By inductively applying Theorem 12 on the length of the trace n we can conclude that all processes in $\mathscr{P}_n$ type in the S_n.

In particular, the process $\mathbf{out}(M, N); P'$ types in state S_n and hence $\{\!|\rho_n(\mathsf{msg}(M, N))|\!\}_{S_n} \in \mathscr{F}_{P_0}$.

Theorem 7 establishes the final relation between the inference system of Figure 6.10 and the instrumented semantics. We use this result to link the facts generated by the translation to a query of interest.

COROLLARY 1 (CHECKING QUERIES) *If* $\rho_0, S_0, \mathscr{P}_0 = \{(P_0 \mid A, \emptyset, \emptyset)\} \rightarrow^* \rho_j, S_j, P_j = P'_j \uplus \{(\mathbf{out}(M,N); P', L, V)\}$
$\rightarrow^* \rho_n, S_n, P_n$ *and* $\rho_n, S_n \models b$ *then there exists an* $\alpha_b \in restrict(\alpha_0, b)$ *where* $\theta = mgu(S_n, \alpha_b)$ *and* $\{\!|\rho_n(\mathsf{msg}(M,N))|\!\}_{\theta \circ \alpha_b}$.

PROOF.Follows from Theorem 7 and because $S_j \preceq_\emptyset S_n$.

Therefore we can express any query of the form:

$$\mathsf{msg}(M,N) \text{ where } b$$

where b is a boolean expression ranging over names and monadic constructors in M and N. Queries of this form are general enough to model secrecy from the attacker's perspective (assuming that the channel M is public), as well as the authentication properties discussed in Section 6.5.

6.8 Experimental Evaluation

We implemented our analysis into a prototype tool written in Haskell, that translates processes specified in Set-Pi and uses the ProVerif as a back-end resolution engine for Horn clauses.

Figure 6.11 shows the results for our examples: the running example on CANAuth, a flawed version of MaCAN [BSNN14], a key registration protocol, and an implementation of the Yubikey protocol modeled after [KK14]. We recorded the running times for our test suite on a 2,7 GHz Intel Core i7 with 8 GB of RAM running OS X. They are comparable to similar ProVerif models in applied-π, which shows that there is little overhead induced by our specific translation.

6.9 Conclusions and Related Work

The Set-Pi calculus and its set-based abstraction method provide an important step to overcome a serious limitation in current automated protocol verification: the limited support verifying protocols that use state. When a change in state can lead to non-monotonicity (things that were possible before the change are not possible after it) then

Example	Average time	Vulnerable
CANAuth	0.0174s	no
MaCAN	0.0244s	yes
Key registration	0.0254s	no
Yubikey single	0.0194s	no

Figure 6.11: Experimental results

the standard abstraction and resolution approach leads to false positives. Our solution is to enter just the "right amount" of state information into the abstraction of messages: enough to represent the non-monotonic aspects we want to model, but only so much that we do not destroy the benefit of the stateless abstraction approach in the first place. This work has been inspired by several works that go in a similar direction and we discuss here how they relate to us.

The closest works are two articles that similar to this work add state-information into abstraction-based approaches. The AIF framework [Möd10] first used the idea of encoding set memberships into the state abstraction. AIF is based on the low-level AVISPA Intermediate Format [The03] and thus does not have the declarativity of a process calculus. For instance, one has to explicitly specify the attacker and cannot derive it from the calculus. Further, AIF uses the "raw" set membership abstraction, while in our abstraction approach we do integrate the context in which messages have been created which gives a finer abstraction. Also Set-Pi uses locks on sets, while AIF does not have this notion (and the lock exists only as per the scope of each AIF transition rule).

The second similar stateful abstraction approach is StatVerif [ARR11] which also provides an extension of the applied π calculus. While we use state-information in the abstraction of messages, StatVerif encodes state-information as an additional argument in the generated predicates of the Horn clauses. The state transition that this approach supports are in some sense like "breaking glass": we can make at some point a global change which cannot be reverted (to avoid cycles in the state transition graph). We believe that Set-Pi and StatVerif have some complementary strengths as there are examples that cannot be directly expressed in the other. While StatVerif can express that a set of messages makes a state-transition at the same time, our abstraction looses this relation between messages. On the other hand, we can flexibly have messages change their set membership independent of each other, and they can return to any previous state. An argument for the expressiveness of Set-Pi is that we have a systematic way to formalize agreement properties using sets.

There are several model-checking approaches that can deal with stateful protocols, namely the AVISPA/AVANTSSAR platform [AAA+12]. Note that here one needs to

bound the number of steps of honest agents which is often fine for finding attacks, but gives limited guarantees for verification. In fact, [FS09] studies APIs of key tokens using SATMC [AC08] of AVISPA, and considers abstractions of data similar to our set abstraction. This can in some cases lead to finitely many reachable abstracted states so the analysis despite depth bound is complete. The AVANTSSAR platform also includes the novel specification language ASLan that besides sets also supports the formulation of Horn-clause policies that are freshly evaluated in every state. For certain fragments we can obtain effective model-checking approaches, but again at the price of bounding the number of steps of honest agents.

The work [KK14] presents a verification approach for stateful protocols using the Tamarin prover [SMCB13] that is based on backwards analysis using multi-set rewriting. This is not fully automated and often requires the user to supply auxiliary lemmas tailored to the problem. The benefit of this approach is a great amount of flexibility in formalizing relationships between data that cannot be captured by a particular abstraction and resolution approach.

CHAPTER 7

Proving Stateful Injective Agreement with Refinement Types

As we have seen in Chapter 4, embedded protocols require a certain amount of state to operate. In traditional Internet-oriented communication protocols, injective agreement between two parties is ensured with challenge-response mechanisms: a fresh nonce is sent to be signed along with a response, to be later checked. Instead, embedded protocols must often deal with restrictions that hinder the applicability of a challenge-response pattern, as we have shown in Chapter 4 for our case study. Therefore protecting from replay attacks must be handled with internal state. For example, in CANAuth this consists of a counter, of which a copy is kept by the two communicating parties, and is increased for every new message sent.

This pattern is also present in mobile GSM networks [AMRR11] to avoid unnecessary roundtrips during authentication, and was also proposed as an optional feature of emerging Internet protocols like QUIC [FG14, LJBNR15] that support a zero round-trip mode. There, a so-called "strike-register" could be used to ensure freshness of short-lived nonce received by the server until a time-stamp invalidates the nonce.

In Chapter 6 we presented Set-Pi, an extension of the Applied Pi-calculus that coupled with an encoding into Horn clauses that maintains enough state information so that

non-monotonic behaviour could be analysed by monotonic, saturation-based solvers, such as ProVerif. The key idea there is to abstract values into their set-membership class, and to use this state information as a pattern upon which certain actions are taken. We showed how events are converted into set transitions and injective-agreement properties into reachability queries of a specific membership class for the nonces identifying the session.

While analysing protocols at an abstract level — such as with the Applied Pi-calculus — is useful to reason about the correctness of a protocol design, there is an abstraction gap between Applied Pi-calculus models and protocol specifications. In particular Set-Pi defines a notion of sets that can be used to model a wide range of concrete mechanisms, such as counters, timestamps, databases of keys and nonces. In this chapter we aim at closing this gap by constructing a verified implementation of a protocol using these concrete mechanisms.

Contributions We present a novel way of proving injective agreement properties directly on the executable specification of the protocol extending previous work on verifying weak agreement using event logs [FKS11]. We prove these properties using refinement types — types with attached logic formulas that express properties on data — and the F* language [SHK+15] — an ML-like language with support for refinement types. The key idea is to maintain a log of events in the protocol, and to type the insertion of an event in the log such that the desired injective correspondence property is preserved.

In line with Bellare and Rogaway's game playing technique [Sho04, BR04], the proofs that we generate consist of a sequence of games $\mathcal{G}_1 \rightarrow \cdots \rightarrow \mathcal{G}_n$, where the original game $\mathcal{G}_1$ corresponds to the protocol code, and successive transformations $\mathcal{G}_i \rightarrow \mathcal{G}_{i+1}$ are indistinguishable up to a negligible probability. The final game $\mathcal{G}_n$ is one such that the security property can be easily verified, hence typing succeeds.

Our approach differs from previous work using refinement types and affine logic [BCEM15] in that we use classical logic formulas on the event traces to prove the strong agreement property. This allows us to use F* — an extension of the F# functional language with refinement types, supported by an SMT-based type checker — to obtain a provably correct implementation of the protocol.

7.1 Review: Computational RCF

First we review Computational RCF, the core calculus used by F*, which is an extension of RCF with probabilistic semantics, and no non-determinism. For a complete and

a,b,c		label
x,y,z		variable
φ		first-order logic formulas
$h ::=$		value constructor
	$inl \mid inr$	left/right constructor of sum type
	$fold$	constructor of recursive type
$M,N ::=$		value
	$\mid x \mid () \mid \textbf{fun}\ x \to A$	variable, unit, function
	$\mid (M,N) \mid h\ M$	pair, construction
	$\mid \textbf{read}_a \mid \textbf{write}_a$	reference reader/writer
$A,B ::=$		expression
	$\mid M \mid M\ N$	value, application
	$\mid \textbf{let}\ x = A\ \textbf{in}\ B \mid \textbf{let}\ (x,y) = M\ \textbf{in}\ A$	let binding, pair split
	$\mid \textbf{match}\ M\ \textbf{with}\ h\ x \to A\ \textbf{else}\ B$	constructor match
	$\mid \textbf{assume}\ \varphi \mid \textbf{assert}\ \varphi$	assumption/assertion of φ
	$\mid \textbf{sample} \mid \textbf{ref}\ M$	fair coin toss, reference creation

$\textbf{true} \triangleq inl\ ()$ $\quad$ $\textbf{false} \triangleq inr\ ()$

Figure 7.1: Syntax of Computational RCF

formal treatment we refer to the original article [FKS11]. A probabilistic semantics allows precise modelling of probabilistic cryptographic algorithm and adversaries. As non-determinism in practice gives the adversary the power of an NP-oracle, the semantics of the calculus excludes it. To prove computational soundness of the cryptographic primitives, we instead rely on the ability of the attacker to call the protocol.

Figure 7.1 presents the core calculus. It is a simplified version of F*, which will be used throughout this chapter, however, the extensions that we use are standard ML syntactic sugar on top of RCF.

The syntax contains the standard categories of algebraic constructors, values and expressions. Semantic rules are expressed in the form $[X,L,A] \to_p [X',L',B]$ where A is an expression that reduces to B in one step with probability p, changing the environment X into X', and the set of logical assumptions L into L'. The additions to the calculus include **assume** and **assert**. The semantics for **assume** φ introduces a new hypothesis φ to a list hypothesis in the program, and **assert** φ introduces a proof obligation, requiring φ to hold when the **assert** is executed. The formula φ can be any first-order logic

$T,U,V ::=$		types
	$\mid \alpha \mid \mathit{unit}$	type variable, unit
	$\mid x:T \to E\ U \mid x:T * U$	dependent function type, dependent pair type
	$\mid T+U \mid \mu\alpha.T$	disjoint sum type, recursive type
	$\mid x:T\{\varphi\}$	refinement type
$E ::=$		effects
	$\mid$ **Tot** $\mid$ **Div** $\mid$ **St**	total, divergent, stateful functions
	$\mid$ **Exn** $\mid$ **All**	functions with exception, all effects

$\mathit{bool} \triangleq \mathit{unit} + \mathit{unit} \qquad \mathbf{ref}\ T \triangleq (() \to T) * (T \to ())$

Figure 7.2: Refinement types

formula such that fv φ are bound in the current environment. In F*, φ can be any total Boolean expression.

The expression **sample** allows sampling one bit of information with probability $\frac{1}{2}$, and returns either **true** or **false**. This allows to express the probabilistic behaviour used in cryptography.

References are in ML-style, and can be used for programming stateful computation, including stateful protocols, communication buffers, and oracles. They are represented as pairs of functions that read and write in a specific memory location, and each execution of **ref** creates a fresh label a, and returns a pair of functions $(\mathbf{read}_a, \mathbf{write}_a)$ that read and write on a, respectively. Hence we can write !**M** for dereferencing a memory location, and **M** := N for assigning N to the location **M**.

Runtime Safety An RCF program A is *safe* if for every possible evaluation of the program all assertions succeed. Hence whenever a formula is evaluated, this formula is a consequence of the previously evaluated formulas.

As the RCF programs are probabilistic, one can define *probabilistic runtime safety* of A as the limit with $n \to \infty$ of the sum over all n-step probabilistic evaulations $A \to_{p_1} \cdots \to_{p_n} A_n$ of $p_1 \cdot \ldots \cdot p_n$ in which all assertions succeed. Probabilistic runtime safety computes the total probability over all possible executions that the assertions succeed: this is a useful concept that we will later use for defining our cryptographic properties.

Type Safety Figure 7.2 presents the type system that we use in our programs. Again, we refer to the original article for a fully formal treatment [FKS11]. As we have seen, refinements are formulas applied on data, and are denoted by the type $x : T\{\varphi\}$. The typing judgement holds for every element x of type T for which $\varphi(x)$ holds. Note that dependent function types are annotated with an *effect system*. The elements of the effect system compose a lattice with the order relation **Tot** $<$ **Div** $<$ **St** $<$ **All** and **Div** $<$ **Exn** $<$ **All**. Here **Tot** denotes a pure and terminating function, **Div** denotes a computation that can diverge, **St** denotes a stateful function, **Exn** denotes a function that can raise an exception, and **All** denotes a function that can have the combination of all other effects. F* allows treating any pure terminating function with Boolean return type as a logic formula, unifying the language of refinements and code.

Type judgements are of the form $I \vdash A : T$, where: I is a type environment, that is a mapping from symbols to types, A is an expression closed under I, and T is a type judgement for A under the type environment I. An alternative typing judgement is $I \vdash B \rightsquigarrow I'$ to express that A exports an interface I' under assuming the environment I.

An important typing result is *compositionality*: if $I \vdash A \rightsquigarrow I'$ and $I' \vdash B : T$ then also $I \vdash A \cdot B : T$. Hence the type system allows to compose two RCF expressions A and B, typed independently, and type the composition accordingly.

Refinement types give strong guarantees about the runtime behaviour of RCF programs. The *type safety* results ensure that if $\emptyset \vdash A : T$, then A is *safe*. That is, if A types with T under an empty set of assumptions, then the formulas enforced by T and the **assert** are never violated at runtime.

7.2 Stateful Authentication

So far, we have reviewed existing literature. Now we will build on it in order to prove injectivity. We present here our running example, a simple protocol that uses state in order to ensure injective agreement between two parties, A and B:

$$A \to B : m, c, hmac(k, \langle m, c \rangle)$$

When A wants to send an authenticated message m to B, it creates a fresh challenge c and signs the m with c, using the key k shared between A and B. Upon receiving the message, B checks the challenge c against its own local state, and the validity of the signature. If c is a fresh value and the *hmac* signature is valid, then the message is accepted, otherwise it is rejected. The challenge can be implemented with multiple mechanisms, including counters, timestamps, and tracking random nonces. We only require that there is a way to check their freshness.

Implementation We implement the skeleton of the protocol C^{PR} with two processes *proc_a* and *proc_b*, shown in Figure 7.3. Both principals *proc_a* and *proc_b* share a session key *k*, which is generated once and for all in the protocol with an attached property *msg_prop*, which specifies the security property, as we will see later. The *proc_a* encodes the message into a tagged bitstring using the function *Format.message*, produces its signature, and finally sends the assembled message. The *proc_b* disassembles it, checking all the conditions that violate the property. Such conditions are signaled by the use of the *option* return type.

```
logic type Message : text → uint32 → Type
logic type msg_prop (msg:text) = (∃ m c. msg = Format.message m c ∧ Message m c)

let k = keygen msg_prop

let proc_a m =
  let c = fresh_challenge () in
  assume (Message m c);
  let msg = Format.message m c in
  let tag = mac k t in
  send (append msg tag);
  None

let proc_b () =
  let msg_tag = recv () in
  if length msg_tag = msg_size + mac_size then (
  let (msg, tag) = split msg_tag msg_size in
  match Format.parse msg with
  | Some (m, c) →
      if is_fresh_challenge c then (
        if verify k m tag then (
          recv_lemma m c !log_pr;
          log_and_update m c;
          assert (Message m c);
          None
        ) else Some "MAC verification failed"
      ) else Some "Challenge used"
  | None → Some "Bad tag"
  ) else Some "Wrong length"
```

Figure 7.3: Protocol skeleton for stateful authentication

The function *fresh_challenge* creates a fresh challenge to be used by *proc_a*, the function *is_fresh_challenge* returns true if the challenge has not been observed by *proc_b*, and

log_and_update adds the challenge to *proc_b*'s local state. Communication between the two parties is done using the *send* and *recv* functions, while the cryptographic component is handled by the *MAC* module.

7.3 Verification Approach

We break down the problem of verifying injective agreement in two steps. First we prove *weak agreement* as shown in the RPC protocol [FKS11]: a custom predicate on the signatures ensures that whenever the *proc_b* accepts a message, the message has previously been produced and sent by an honest principal. Then we use the log of events to prove *strong agreement*: whenever we insert an event in the log that corresponds to the acceptance of the message by the *proc_b*, we ensure by typing that the event does not appear in the log.

The combined use of **assume** and **assert** with the predicate *Message m c* proves weak agreement by typing: the protocol types if, whenever *Message m c* is asserted, it was previously assumed as an hypothesis. Similarly the typing of *log_and_update m c* proves strong agreement, as its type requires the event *Recv m c* not to be in the log. We use refinement types to attach these properties on data, and to prove invariants on the structure of the log.

7.3.1 Cryptographic Verification of MAC

Recalling the INT-CMA game described in Section 3.4, a secure MAC signature is one such that an attacker, having access to an oracle to compute an arbitrary number of signatures for a set of plaintexts T, can forge a signature for a plaintext $t \notin T$ with negligible probability.

Here we show how to express the INT-CMA assumption in the type system. We define a logic type *key_prop* that is used to attach a property to the key and the text, a type constructor *pkey*, a type *entry* for log entries, and a log *log_m*.

type *key_prop* : *key* → *text* → *Type*
type *pkey* (*p*:(*text* → *Type*)) = *k*:*key*{*key_prop k* == *p*}
type *entry* = | *Entry* : *k*:*key* → *t*:*text*{*key_prop k t*} → *m*:*tag* → *entry*
let *log_m* = **ref** []

Intuitively, the property *key_prop k t* can be read as "the oracle has generated the MAC tag of the plaintext *t* using the key *k*". Hence, as long as the property holds, the attacker

has not forged a valid MAC. The *Entry* constructor requires *key_prop* to hold on k and t before the entry can be created. Finally, we define a log that is used for checking our cryptographic assumption.

Ideal Interface I The ideal interface of MAC encodes the properties of the INT-CMA game in the type signatures. Our type signatures have been adapted from [FKS11] to fit in the state monad *St*, which we require for verifying our protocol. The function *mac* requires that the attached *key_prop* holds on k and t, while *verify* ensures that *key_prop* holds if the verification succeeds.

val *keygen*: p:(*text* → *Type*) → *pkey p*
val *mac*: k:*key* → t:*text*{*key_prop k t*} → *St tag* (**modifies** !{*log_m*})
val *verify*: k:*key* → t:*text* → *tag* → *St* (b:*bool*{b ⟹ *key_prop k t*}) (**modifies** !{*no_refs*})

An attacker that is able to win the INT-CMA game can construct a MAC tag *tag* for the combination of k and t without requiring *key_prop k t* to hold. Note that this would violate the type of *verify*, hence no such well-typed attacker exists.

The property *msg_prop* of Figure 7.3 requires that the signed message is tagged and distinct from others, and that *Message m c* has been assumed, therefore the MAC is not forged by an adversary. This property is attached to the key k when it is generated. Typing the protocol code under this interface ensures the weak agreement property.

Ideal Implementation C^I The ideal implementation of MAC maintains a log of all the entries *Entry k t* on which *mac* is called. Calling *verify k t m* succeeds only if the tag m is the result of computing *mac k t and* if *Entry k t* is present in the log.

```
let keygen (p: (text → Type)) =
  let k = sample keysize in assume (key_prop k == p); k
let mac k t =
  let m = hmac sha1 k t in log_m := Entry k t m :: !log_m; m
let verify k t m =
  let verified = (m = hmac sha1 k t) in
  let found = is_Some (find (fun (Entry k' t' m') → k=k' && t=t') !log_m) in
  verified && found
```

The ideal implementation C^I types under the interface I, hence we have $\emptyset \vdash C^I \rightsquigarrow I$.

Concrete Implementation C^C The concrete implementation C^C differs from the ideal implementation C^I in that *verify* does not check the presence of k and t in the log.

```
let keygen (p: (text → Type)) =
  let k = sample keysize in assume (key_prop k == p); k
let mac k t =
  let m = hmac sha1 k t in log_m := Entry k t m :: !log_m; m
let verify k t m =
  m = hmac sha1 k t
```

By the INT-CMA assumption on MAC the attacker can forge a signature that successfully verifies only with negligible probability, hence we have that $C^I \approx C^C$.

7.3.2 Stateful Injectivity

The protocol presented in Figure 7.3 relies on state to ensure an injective correspondence between the two processes. Here we present the three mechanisms that we identified in the various protocols: counters, timestamps and Bloom filters.

As said at the beginning of the section, we maintain injectivity by typing the function *log_and_update* that updates the server state and a log of events of the form *Recv m c*, which indicate that *B* has received and accepted a message.

```
val log_and_update: m: text → c: uint32 → St (unit)
  (requires (fun h → Invariant h ∧ PreLogUpd m c (sel h log_p) ∧
                     ∀e . List.mem e (sel h log_p) ⟹ e <> (Recv m c)))
  (ensures (fun h x h' → Invariant h' ∧ PostLogUpd m c (sel h log_p) ∧
                         sel h' log_p = Recv m c :: sel h log_p))
  (modifies !{log_p})
let log_and_update m c =
  log_p := Recv m c :: !log_p;
  update_challenge c
```

The type of *log_and_update* requires that *Invariant h* be maintained before and after the execution of the function. The precondition *PreLogUpd m c* (*sel h log_p*) relates the current combination of message *m* and challenge *c* with the state of the log before the execution of the function, and similarly does *PostLogUpd m c* (*sel h' log_p*) relate them after both the log and the server state have been updated. The combination of precondition and the postcondition is dependent on the specific mechanism, which we are soon going to detail. What is maintained over the interfaces for counters, timestamps and Bloom filters is the third precondition, which requires that all events in the log before execution are different from the one being inserted, and the third postcondition, which ensures that the log is properly updated.

Note that *log_and_update* only types when the event *Recv m c* is not in the log, hence it can insert the event only once: each successive call with the same parameters violates the typing after the state update. Next we are going to detail the specific invariants and the functionality of the three stateful mechanisms.

Counters St^C As previously mentioned we ensure strong authentication by typing the *log_and_update* function:

```
logic type Invariant (h:heap) =
  max_challenge (sel h log_p) = sel h proc_b_st ∧ Heap.contains h proc_b_st
  ∧ Heap.contains h proc_a_st ∧ Heap.contains h log_p ∧ proc_b_st <> proc_a_st
logic type PreLogUpd (m: text) (c: uint32) (h: heap) =
  c > max_challenge (sel h log_p)
logic type PostLogUpd (m: text) (c: uint32) (h: heap) =
  c = max_challenge (sel h log_p)
```

The *Invariant* ensures that a set of properties are maintained on the heap across executions of *proc_a* and *proc_b*. In particular, we maintain the invariant that the highest counter in *log_p* is equal to the current value of *proc_b_st*, and that the *proc_a_st* and *proc_b_st* are disjoint memory locations, hence *proc_a* and *proc_b* do not interfere.

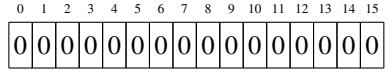
The following code implements the counter interface that is used by the protocol.

```
let proc_a_st = ref 1
let proc_b_st = ref 0
let fresh_challenge () =
  let c = !proc_a_st in
  proc_a_st := c+1; c
let is_fresh_challenge c =
  c > !proc_b_st
let update_challenge c =
  proc_b_st := max c (!proc_b_st)
let check_challenge c st =
  c ≤ st
```

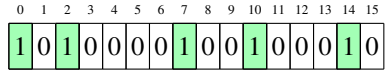
The following lemma is used for typing the *log_and_update* function.

```
val recv_lemma: m:text → c:uint32 → (l:list event{c > max_chal l}) →
  Lemma(∀ e . List.mem e l ⟹ e <> (Recv m c))
```

Intuitively, *recv_lemma* asserts that whenever we receive a message *m* that is signed with a counter *c* greater than all the counters in the log, then the event *Recv m c* is not in the log.



(a) Empty bloom filter consisting of 16 bits.



(b) Bloom filter containing "hi", "hola", "hello". Here we assume $hash^1(\text{hi}) = 7$, $hash^2(\text{hi}) = 14$, $hash^1(\text{hola}) = 10$, $hash^2(\text{hola}) = 2$, $hash^1(\text{hello}) = 7$, and $hash^2(\text{hello}) = 0$.

Figure 7.4: Example of Bloom filter

Timestamps St^T Timestamps are very similar to counters for the sake of proving injective agreement. In fact, our stateful module for timestamps only differs in the definition of *fresh_challenge* and in the addition of the *tick* function.

let *fresh_challenge* () = !*proc_a_st*
let *tick* () = *proc_a_st* := (!*proc_a_st*) + 1

Bloom filters St^B Bloom filters are an efficient data structure that can track, with a fixed number of bits, an arbitrary number of values in a set. Bloom filters trade efficiency for precision, in the sense that a membership query gives either a negative answer — i.e., a certain value is not in the set — or a probabilistic answer — i.e., the value may be in the set with a certain probability. Hence they are considered a *probabilistic data structure*.

The use of Bloom filters is very appealable for the construction of efficient security protocols, since they can be employed to efficiently track values such as random numbers and keys instead of relying on multiple, time-consuming message exchanges. One interesting example is the QUIC protocol [FG14, LJBNR15] with support for a zero round-trip mode: a so-called "strike-register" is used to ensure freshness of short-lived nonces received by the server until a time-stamp invalidates the nonce.

The data structure consists of a bitstring of a fixed length *ln*, like the one in Figure 7.4 (a). Adding an element to the Bloom filter requires computing *n* different hashes of the data, then setting bits of the bitstring with index corresponding to those hashes. Checking whether an element is present in the filter requires re-computing the *n* different hashes for the data, then checking that all bits with index corresponding to those hashes are set to 1.

The example of Figure 7.4 (b) shows a filter where the bits with indexes 0, 2, 7, 10 and 14 are set, and two hashes are computed for each element inserted. Checking whether

one element in $S = \{\text{hi}, \text{hola}, \text{hello}\}$ is in the filter always succeeds, as the hash functions are deterministic. Checking whether an element outside S is in the filter *may* succeed. The check succeeds if the indexes computed by the hash functions map to bits in the filter that are set to one.

Assuming that the hash function in use by the Bloom filter has uniform distribution over its codomain[1], one the rate of false positives of a Bloom filter is $(1 - e^{k \cdot n/ln})^k$, where the parameter k represents the number of computed hashes, the parameter *ln* represents the length of the Bloom filter, and n are the elements being inserted [MU05].

We constructed a module that implements the following interface for Bloom filters:

```
type ln_t = ln:uint16{ln > 0}
type hash = h:seq byte{Seq.length h ≥ 2}
type hash_fn = text → Tot hash
type bloom (ln:ln_t) = bl:seq bool{Seq.length bl = ln}

logic type Le (ln:ln_t) (bl1:bloom ln) (bl2:bloom ln) =
  ∀i . (0 ≤ i && i < ln) ⟹ ((Seq.index bl1 i) ⟹ (Seq.index bl2 i))

val create: ln:ln_t → Tot (bl:bloom ln)
let create ln = Seq.create ln false

val check: ln:ln_t → h:hash_fn → k:nat → t:text → bl:bloom ln → Tot (b:bool)
val add: ln:ln_t → h:hash_fn → k:nat → t:text → bl:bloom ln →
  Tot (bl':bloom ln{Le ln bl bl' ∧ check ln h k t bl'})
```

First, we define a length type *ln_t* as a positive 16-bit integer. The type of hashes is a byte sequence containing at least two bytes that are used as index over the Bloom filter, and a hash function is any total function from a plaintext to a hash value. A Bloom filter with length *ln* is then a sequence of *ln* Boolean values.

Our module for Bloom filters is *salted*: we use one hash function h and hash k times the plaintext t, by prepending a 2-byte representation of an integer i ranging from 0 to $k-1$. Another option would be to use k different hash functions as parameters. The latter choice would not change the theoretical result, however, the specific choice of hash functions may influence the probabilistic performance.

We define a partial order relation *Le* between two Bloom filters *bl1* and *bl2* of length *ln*. The relation holds if, for every bit with index i that is set in *bl1*, the bit with index i is

[1] The hash functions used by Bloom filters do not need to be collision-resistant, even when Bloom filters are used to track nonces in cryptographic protocols. Finding a collision in our case would *block* the attacker, because the protocol would reject nonces that have been recorded in the filter.

also set in *bl2*. Note that this relation defines a lattice of Bloom filters, where the empty filter is at the bottom, and the filter with all bits set is at the top.

The function *create* constructs an empty filter, which is a sequence of zeroes. The function *check* evaluates the membership of *t* in the filter *bl*, returning **true** if all the corresponding bits in the filter are set. Finally the function *add* inserts *t* in the bloom filter *bl*, returning a new filter *bl*'.

Next we define the invariant to be maintained across executions of *proc_a* and *proc_b*, and the pre and post condition to the *log_and_update* function.

```
logic type Invariant (h:heap) =
  Heap.contains h proc_b_st
logic type PreLogUpd (m: text) (c: uint32) (h: heap) =
  true
logic type PostLogUpd (m: text) (c: uint32) (h: heap) =
  check_challenge m c (sel h proc_b_st)
```

This interface is much simpler than the ones used for counter, as one does not need to maintain the relation between the state of the two processes. The interface for Bloom filters that is used by the protocol is as follows:

```
let fresh_challenge () =
  sample num_bits
let is_fresh_challenge c =
  not (Bloom.check c (!proc_b_st))
let update_challenge c =
  proc_b_st := Bloom.add c (!proc_b_st)
let check_challenge c st =
  Bloom.check c st
```

Finally, we present the lemma used to type the *log_and_update* function.

```
val recv_lemma: m:text → c:uint32 →
  bl:bloom filter_size{not (Bloom.check (uint32_to_bytes c) bl)} →
  l:list event{∀ m' c' . List.mem (Recv m' c') l ⟹ Bloom.check (uint32_to_bytes c) bl} →
  Lemma (∀ e . List.mem e l ⟹ e <> (Recv m c))
```

Essentially, *recv_lemma* states that if a Bloom filter *bl* has recorded all the challenges in a log of events *l*, and the new challenge *c* is not stored in *bl*, then the event *Recv m c* is not present in *l*.

7.3.3 Correctness

By composing the modules that we presented throughout this chapter, we obtain the typability of the system from the following theorem.

THEOREM 8 (INJECTIVE AGREEMENT OF C^{Pr}) *Let $St \in \{St^C, St^T, St^{Bl}\}$ be one of the stateful mechanisms. If $I^{MAC} \vdash \mathsf{St} \cdot C^{Pr} : T$, then the composition $C^C \cdot St \cdot C^{Pr}$ of the concrete MAC implementation C^C, preserves injective agreement.*

PROOF. Because the composition of the protocol code and the stateful mechanism satisfies the judgement $I^{MAC} \vdash St \cdot C^{Pr} : T$, and because $\emptyset \vdash C^I \rightsquigarrow I^{MAC}$, then by compositionality we obtain $\emptyset \vdash C^I \cdot St \cdot C^{Pr} : T$.

We can augment the semantics of F* programs by recording traces of events, as follows:

$$[X, L, (\textbf{assume}(\mathit{Message\ m\ c}); A)] \xrightarrow{\mathit{Message\ m\ c}}_1 [X, (\{\mathit{Message\ m\ c}\} \cup L), A]$$

$$[X, L, (\textbf{assert}(\mathit{Message\ m\ c}); A)] \xrightarrow{\overline{\mathit{Message\ m\ c}}}_1 [X, L, A] \qquad \text{if } L \vdash \mathit{Message\ m\ c}$$

$$[X, L, (\mathit{log\_and\_update\ m\ c}; A)] \xrightarrow{\mathit{Recv\ m\ c}}_1 [X', L, A]$$

Where in the last rule *X'* updates the state *X* by appending the event *Recv m c* and tracking the challenge *c*.

Typing the protocol ensures the following two results:

(1) *Agreement* between the events $\overline{\mathit{Message\ m\ c}}$ and *Message m c*. If $\overline{\mathit{Message\ m\ c}}$ occurs in a trace at index *i* of the sequence of transitions, then *Message m c* occurs at index *j* with $j < i$, because the predicate *Message m c* must be assumed before the corresponding assertion is executed.

(2) *Pairwise disjointness* of the events of the form *Recv m c*. Typing an instance of *log_and_update m c* requires that *Recv m c* to be different from all other entries in *log_p*. Hence a specific instance *Recv m c* can only be inserted once.

Because **assert** (*Message m c*) appears only after *log_and_update m c* in C^{Pr}, in order to observe $\overline{\mathit{Message\ m\ c}}$, one can only observe it after *Recv m c* in the sequence of events:

$$\ldots \xrightarrow{\mathit{Recv\ m\ c}}_1 \cdot \xrightarrow{\overline{\mathit{Message\ m\ c}}}_1 \ldots$$

And since *Recv m c* can appear at most once in a trace then also $\overline{\mathit{Message\ m\ c}}$ appears at most once, and only after an occurrence of *Message m c*. Hence injective agreement holds in the composition $C^I \cdot St \cdot C^{Pr}$.

Because $C^I \approx C^C$, then by compositionality $C^I \cdot St \cdot C^{Pr} \approx C^C \cdot St \cdot C^{Pr}$, hence an attacker that interacts with the concrete protocol break injective agreement only up to a negligible probability.

7.4 Conclusions

We showed how to prove injective agreement on a simple stateful protocol that uses state for replay protection. This technique is general enough to be directly applicable to protocols that use various form of stateful tracking mechanisms, and we have seen that the same protocol structure types with counters, timestamps, and random challenges tracked by Bloom filters.

The stateful nature of F* makes it natural to express stateful security protocols. Its support for refinement types allows formal reasoning about their security properties, and the modularity of the language allows to construct libraries of formally verified security mechanisms. Furthermore, the programs that we presented can be compiled into executable code: simply by erasing the refinement formulas and effect annotations one valid obtains F# code.

On the other hand, the use of classical logic with combinations of assumes and asserts required custom manual reasoning about the final result, as we have seen in Theorem 8. We believe that one can overcome this limitation by using logs of events for the sender and receiver processes, and check multiset inclusion to ensure injectivity. Work in this direction has shown that this is achievable by typing, using monotonic references and proving invariants that maintain the inclusion of the two logs [SHK+15].

Another approach is to use affine logic to reason about agreement properties. Affine logic is a custom logic that can naturally express the consumption of resources and replication thereof [Tro92]. Bugliesi et al. [BCEM15] propose an affine refinement type system for RCF to prove resource authorisation properties in cryptographic protocols. The advantage of this approach is that agreement properties are automatically captured by the logic, and don't need custom reasoning. On the other side it gives a great deal of flexibility and support by standard tools such as F*, which in turn relies on the Z3 theorem prover. Supporting custom logics is known to be hard.

Relating to the Set-Pi language of Chapter 6, the primitives that we presented are two concrete patterns that are indeed a set-abstraction. As future work it would be interesting to produce a translation from Set-Pi protocols into formally verified F* protocols. This could be done by means of annotations, for example one can define a mapping between a library of Set-Pi types to a library of modules in F*.

CHAPTER 8

Conclusions

In this thesis we have explored how formal protocol verification techniques can be adopted in the domain of embedded systems, and we have improved existing techniques to better cope with the peculiarities of the embedded domain. We have also given an in-depth analysis of two authentication protocols for the CAN bus network, applying formal verification to discover breaches in their design and to prove that our corrections are sound. At beginning of our journey we claimed that:

language based technologies offer a framework to push the boundaries of protocol verification, both in the symbolic and computational models, so as to encompass the verification of features peculiar to embedded systems.

In Section 8.1 we summarise the contributions of this dissertation, in order to elucidate how each of our developments is instrumental in validating our claim. Finally in Section 8.2 we will discuss the challenges in the analysis of stateful protocols, particularly in the embedded domain.

8.1 Contributions

We shall now briefly review our contributions on the verification of security protocols, particularly in the domain of embedded systems.

Formally verified embedded automotive protocols. We studied the panorama of existing embedded authentication protocols in automotive, identified two proposals that are likely to be good candidates for future adoption, and showed that formal verification can help to improve their security. In-depth analysis of the systems revealed a range of safety and security flaws. Combining formal protocol analysis with schedulability considerations has helped us harden the system design, by showing which security assumptions can be supported by the safety mechanisms, and viceversa, by constructing security protocols that are more suitable for the safety constraints.

Stateful protocols can be analysed by means of process algebras. Process algebras — and the Applied Pi-calculus in particular — offer an expressive language framework for describing security protocols, and are supported by well-established and effective symbolic analyses. However, most successful analysis techniques work by coalescing multiple system states into a single set of reachable information. This over-approximation is too coarse when certain security properties (secrecy, authentication) depend on the system's state. We propose an extension of the Applied Pi-calculus to address these problems: Set-Pi extends the well-known process algebra with support for global state. The state consists of sets of values from the process algebra, which can be manipulated by means of insertion and removal operations, and queried for membership of a particular value. We complemented the language with a formal analysis, which is capable of proving secrecy and authentication properties of stateful protocols.

Verified implementations of stateful authentication mechanisms. Verifying protocol implementations in the computational model reduces the abstraction gap in comparison to symbolic proofs. We propose a methodology to prove strong authentication properties on stateful protocols using refinement types and classical first-order logic in an effectful programming language. We show with a concrete example how multiple stateful mechanisms (e.g. counters, timestamps and random nonces paired with Bloom filters) can be plugged into a unified protocol structure, as they offer similar guarantees.

8.2 Future Directions

Among the properties that we have not studied in this thesis, but that are interesting in some embedded domains, we should mention indistinguishability properties, which reduce to observational equivalence of processes. The problem of proving indistinguishability in the Applied Pi-calculus has received a lot of attention [BAF05, DKR07, CD09, CB13], however, we still lack satisfying results for stateful calculi. Preliminary work in this direction has been done by Mancini [Man15], showing limitations in proving equivalences due to the abstractions required by the StatVerif translation. It also shows

the need to reason about these properties in stateful embedded systems, such as mobile telephony protocols, making this extension highly desirable.

The expressive power of our encoding of Set-Pi into Horn clauses is limited in that the analysis gives precise results only when a certain property depends on a single value. When we want to prove properties that depend on state information regarding multiple values, our over-approximation fails to give the expected guarantees. We circumvented this limitation by using a type system for our language that allows to track the state of multiple values that can be constructed from a single seed, e.g. a public/private key pair. However the problem still remains for cases in which these values are freshly generated by independent processes.

Synchronising the state over multiple values is not a problem in the StatVerif encoding of stateful Applied Pi-calculus models into Horn clauses, where the state is modelled as a tuple of values added to the predicates and all state cells can be changed synchronously. The StatVerif encoding encounters other limitations: when trying to express the potentially infinite structures that are natural in Set-Pi the saturation procedure does not terminate.

In our attempts at combining the expressive power of the two calculi we hit limitations that suggest that a finer abstraction might require a more powerful logic than Horn clauses can offer. Negative results in this respect would help to formalise our intuition. Another interesting research direction is to study a possible extension of the underlying logic that provides the required expressive power.

Regarding our work in Chapter 7, our verified implementation of authenticated protocols is hand-crafted, but at the same time it is modular enough that the proofs for the different stateful mechanisms (counters, timestamps, Bloom filters) can be swapped in the same protocol structure.

These stateful mechanisms are concrete implementations of the set abstraction that we employed in Chapter 6. The problem of refining abstract specifications into concrete working implementation poses interesting challenges, and has been addressed before [TH05, CB12, Mod12, QPN14, AM14]. Refining specifications of stateful protocols into implementations — in particular verifiable implementations — is still a rather unexplored and interesting domain.

APPENDIX A

Proofs of Chapter 6

This chapter presents the proofs for Set-Pi, sketched in Chapter 6. Section A.1 shows the subject reduction results for the type system used in the language. Section A.2 shows the soundness of the transformations used to prove agreement and injective agreement in Section 6.5. Section A.3 shows the correctness of the analysis presented in Section 6.6 with respect to the semantics of Section 6.4; these proofs were sketched in Section 6.7.

A.1 Subject Reduction

THEOREM 9 (SUBJECT REDUCTION) *Let Γ be a type environment, $\rho, S, \mathscr{P}$ a configuration. If for all $(P,L,V) \in \mathscr{P}$ we have $L,\Gamma[\widehat{\rho}] \vdash P$, and if $\rho, S, \mathscr{P} \rightarrow \rho', S', \mathscr{P}'$, then for all $(P',L',V') \in \mathscr{P}'$ we have $L',\Gamma[\widehat{\rho'}] \vdash P'$.*

PROOF. We prove the theorem with a case-by-case analysis of the semantic step $\rho, S, \mathscr{P} \rightarrow \rho', S', \mathscr{P}'$.

Case COM.
$\rho, S, \mathscr{P} \uplus \{(\mathbf{in}(M, x:T); P_1, L_1, V_1), (\mathbf{out}(M,N); P_2, L_2, V_2)\} \rightarrow \rho', S, \mathscr{P} \uplus \{(P_1\{^{N'}/_x\}, L_1, N' :: V_1), (P_2, L_2, V_2)\}$, where $\rho' = mgu(N', N) \circ \rho$ and $N' = pt_x^{ri(V)}(T)$.

Let (P,L,V) be a process that remains unchanged after the transition. We need to prove that if $L,\Gamma[\widehat{\rho}] \vdash P$ then $L,\Gamma[\widehat{\rho'}] \vdash P$. We know that N is a ground term, and N' is homomorphic to N by construction, where every instance of a name is mapped to a syntactically unique variable[1]. Hence ρ and ρ' satisfy the properties of Lemma 2, namely that $Dom(\rho) \subseteq Dom(\rho')$ and that for every variable x in ρ we have $\rho'(x) = \rho(x)$. Therefore we conclude that $L,\Gamma[\widehat{\rho'}] \vdash P$.

For P_2 we know by hypothesis that since $L_2,\Gamma[\widehat{\rho}] \vdash \mathbf{out}(M,N); P_2$ then also $L_2,\Gamma[\widehat{\rho}] \vdash P_2$. Applying Lemma 2 we conclude that $L_2,\Gamma[\widehat{\rho'}] \vdash P_2$.

For P_1 we know by hypothesis that $L_1,\Gamma[\widehat{\rho}] \vdash \mathbf{in}(M,x:T); P_1$, and hence $L_1,\Gamma[\widehat{\rho},\ x \mapsto T] \vdash P_1$. By Lemma 2, since x does not appear in $Dom(\Gamma) \cup Dom(\rho')$, we obtain $L_1,\Gamma[\widehat{\rho'},\ x \mapsto T] \vdash P_1$ and by Lemma 1, because $\Gamma[\widehat{\rho'}] \vdash N' : T$ we obtain $L_1,\Gamma[\widehat{\rho'}] \vdash P_1\{^{N'}/_{x}\}$.

Therefore all processes type after the transition.

Case NEW.
$\rho, S, \mathscr{P} \uplus \{(\mathbf{new}^l\ x:a; P,L,V)\} \to \rho, S, \mathscr{P} \uplus \{(P\{^{a^l[V]}/_{x}\},L,V)\}$

By hypothesis $L,\Gamma[\widehat{\rho},x \mapsto a] \vdash P$ and since $\Gamma \vdash a^l[V] : a$ by Lemma 1 we conclude that $L,\Gamma[\widehat{\rho}] \vdash P\{^{a^l[V]}/_{x}\}$. For all processes in $\mathscr{P}$ the typing judgement is unchanged after the transition, hence it holds by hypothesis.

Case REPL.
$\rho, S, \mathscr{P} \uplus \{(!^k P,\emptyset,V)\} \to \rho', S, \mathscr{P} \uplus \{(P,\emptyset,x_k :: V),(!^{k+1} P,\emptyset,V)\}$ where $\rho' = \rho\{^{k}/_{x_k}\}$.

By hypothesis:

$$\frac{\emptyset,\Gamma[\widehat{\rho}] \vdash P}{\emptyset,\Gamma[\widehat{\rho}] \vdash !^k P}$$

By Lemma 2 we conclude

$$\emptyset,\Gamma[\widehat{\rho'}] \vdash P$$

and hence also

$$\emptyset,\Gamma[\widehat{\rho'}] \vdash !^{k+1} P$$

applying the typing rule for replication.

Let (P',L',V') be a process in $\mathscr{P}$, then by hypothesis $L',\Gamma[\widehat{\rho}] \vdash P'$. Since $x_k \notin Dom(\rho)$ and $\rho' = \rho\{^{k}/_{x_k}\}$, by Lemma 2 we conclude $L',\Gamma[\widehat{\rho'}] \vdash P'$.

[1] The use of position indexes in constructors and replication indexes serves this purpose.

Remaining cases.
For all remaining cases we have a semantic rule of the form $\rho, S, \mathscr{P} \uplus \{(P,L,V)\} \rightarrow \rho, S', \mathscr{P} \uplus \{(P',L',V')\}$. We observe that in all these cases $L, \Gamma[\widehat{\rho}] \vdash P$ contains $L', \Gamma[\widehat{\rho}] \vdash P'$ as hypothesis, hence it holds after the transition. For all processes in $\mathscr{P}$ the typing judgement is unchanged after the transition, hence it holds by hypothesis.

A.2 Agreement

THEOREM 10 *Let P be an extended process with events. If there is no reachable state S from $P' = agree(P)$ that satisfies the expression $M \in e_2 \wedge \neg M \in e_1$, then there is an non-injective agreement between $e_1(M)$ and $e_2(M)$ in P.*

PROOF.One can construct a simulation relation between $agree(P)$ and P, defined as a binary relation on semantic configurations: $\rho_1, S_1, \mathscr{P}_1 \propto \rho_2, S_2, \mathscr{P}_2$.

In particular, when $P = \mathbf{event}\ e(M); P_1$ we have that if $(3,1) \in \propto$ and:

${}^1\rho, S, \mathscr{P} \uplus \{(\mathbf{event}\ e(M); P_1, L, V)\} \stackrel{e(M)}{\rightarrow}$

${}^2\rho, S, \mathscr{P} \uplus \{(P_1, L, V)\}$

then:

${}^3\rho, S', \mathscr{P}' \uplus \{(\mathbf{lock}(e); \mathbf{set}(M \in e); \mathbf{unlock}(e); P_2, L, V)\} \rightarrow$

${}^4\rho, S', \mathscr{P}' \uplus \{(\mathbf{set}(M \in e); \mathbf{unlock}(e); P_2, L, V)\} \rightarrow$

${}^5\rho, S' \cup \{(e,M)\}, \mathscr{P}' \uplus \{(\mathbf{unlock}(e); P_2, L, V)\} \rightarrow$

${}^6\rho, S' \cup \{(e,M)\}, \mathscr{P}' \uplus \{(P_2, L, V)\}$

where $P_2 = agree(P_2)$ and $(6,2) \in \propto$.

When a set operation precedes an event, that is:

$$\mathbf{set}(b^+); \mathbf{event}\ e(M); P_1$$

the set transition and the event are merged into a single set transition in the transformed process:

$$\mathbf{lock}(e); \mathbf{set}(b^+; M \in e); \mathbf{unlock}(e); P_2$$

Hence we build the following simulation, where the third component extends the relation with the event that the process being simulated has still to emit, ε indicating that there is no pending event in the simulated process. If $(4,1,\varepsilon) \in \propto$ and:

${}^1\rho, S, \mathscr{P} \uplus \{(\mathbf{set}(b^+); \mathbf{event}\ e(M); P_1, L, V)\} \rightarrow$

$^2\rho, S', \mathscr{P} \uplus \{(\textbf{event}\ e(M); P_1, L, V)\} \overset{e(M)}{\rightarrow}$
$^3\rho, S', \mathscr{P} \uplus \{(P_1, L, V)\}$

then:

$^4\rho, S'', \mathscr{P}' \uplus \{(\textbf{lock}(e); \textbf{set}(M \in e); \textbf{unlock}(e); P_2, L, V)\} \rightarrow$
$^5\rho, S'', \mathscr{P}' \uplus \{(\textbf{set}(b^+; M \in e); \textbf{unlock}(e); P_2, L, V)\} \rightarrow$
$^6\rho, S''', \mathscr{P}' \uplus \{(\textbf{unlock}(e); P_2, L, V)\} \rightarrow$
$^7\rho, S''', \mathscr{P}' \uplus \{(P_2, L, V)\}$

where $P_2 = agree(P_1)$ and $(7, 2, e(M)) \in \propto$ and $(7, 3, \varepsilon) \in \propto$.

Because P is well-typed all the sets modified by $\textbf{set}(b^+)$ are locked, and there is no unlock operation before the event. Hence for any trace in the original process, where the set transition and the event are not consecutive, there is an equivalent trace where the interleaved transitions execute before the set transition.

In particular if a correspondence between $e_1(M)$ and $e_2(M)$ — where **event** $e_2(M)$ follows a set operation — is violated by a trace where **event** $e_1(M)$ occurs between the execution of the set transition and **event** $e_2(M)$, then there exists a trace where **event** $e_1(M)$ occurs before the set transition, and such trace can be simulated by $agree(P)$.

So if $\rho_0, S_0, \mathscr{P}_0'$ has no trace such that there exists a k where $S_k \models M \in e_2 \wedge M \notin e_1$ then it cannot simulate a trace from $\rho_0, S_0, \mathscr{P}_0$ such that $e_2(M)$ has fired but $e_1(M)$ has not, but since $\rho_0, S_0, \mathscr{P}_0' \propto \rho_0, S_0, \mathscr{P}_0$, there is no trace violating the non-injective agreement.

THEOREM 11 *Let P be an extended process with events. If no reachable state S from $P' = inj\text{-}agree(P)$ satisfies the expression $(M \in e_2 \wedge \neg M \in e_1) \vee (M \in twice\text{-}e_2)$, then the injective agreement between $e_1(M)$ and $e_2(M)$ holds in P.*

PROOF.Similarly to the proof for Theorem 10, we build a simulation, this time between processes $inj\text{-}agree(P)$ and P, where:

In particular we have that if $(1, 3) \in \propto$ and:

$^1\rho, S, \mathscr{P} \uplus (\textbf{event}\ e(M); P_1, L, V) \overset{e(M)}{\rightarrow}$
$^2\rho, S, \mathscr{P} \uplus (P_1, L, V)$

then:

$^3\rho, S', \mathscr{P}' \uplus (\textbf{lock}(e, twice\text{-}e); \textbf{if}\ \neg M \in e\ \textbf{then set}(M \in e);$

$$\begin{aligned}
&\textbf{unlock}(e,\textit{twice-e}); P_2 \textbf{ else set}(M \in \textit{twice-e});\\
&\textbf{unlock}(e,\textit{twice-e}); P_2, L, V) \rightarrow^2\\
{}^4\rho, S', \mathscr{P}' \uplus (&\textbf{if } \neg M \in e \textbf{ then set}(M \in e); \textbf{unlock}(e,\textit{twice-e});\\
&P_2 \textbf{ else set}(M \in \textit{twice-e}); \textbf{unlock}(e,\textit{twice-e});\\
&P_2, L \cup \{e,\textit{twice-e}\}, V) \rightarrow\\
{}^5\rho, S', \mathscr{P}' \uplus (&\textbf{set}(M \in e); \textbf{unlock}(e,\textit{twice-e}); P_2,\\
&L \cup \{e,\textit{twice-e}\}, V) \rightarrow\\
{}^6\rho, S' \cup \{(e,M)\}, \mathscr{P}' \uplus (&\textbf{unlock}(e,\textit{twice-e}); P_2,\\
&L \cup \{e,\textit{twice-e}\}, V) \rightarrow^2\\
{}^7\rho, S' \cup \{(e,M)\}, \mathscr{P}' \uplus &(P_2, L, V)
\end{aligned}$$

if $e(M)$ has not previously fired in the trace, then $(2,7) \in \propto$, or:

$$\begin{aligned}
{}^{5'}\rho, S', \mathscr{P}' \uplus (&\textbf{set}(M \in \textit{twice-e}); \textbf{unlock}(e,\textit{twice-e}); P_2,\\
&L \cup \{e,\textit{twice-e}\}, V) \rightarrow\\
{}^{6'}\rho, S' \cup \{(\textit{twice-e},M)\}, \mathscr{P}' \uplus (&\textbf{unlock}(e,\textit{twice-e}); P_2,\\
&L \cup \{e,\textit{twice-e}\}, V) \rightarrow^2\\
{}^{7'}\rho, S' \cup \{(\textit{twice-e},M)\}, \mathscr{P}' \uplus &(P_2, L, V)
\end{aligned}$$

if $e(M)$ already fired in the trace, then $(2,7') \in \propto$.

If a process is of the form:

$$\textbf{set}(b^+); \textbf{event } e(M); P_1$$

it gets translated into:

$$\begin{aligned}
&\textbf{lock}(e,\textit{twice-e}); \textbf{if } \neg M \in e\\
&\textbf{then set}(b^+; M \in e); \textbf{unlock}(e,\textit{twice-e}); P_2\\
&\textbf{else set}(b^+; M \in e); \textbf{unlock}(e,\textit{twice-e}); P_2
\end{aligned}$$

where $P_2 = \textit{agree}(P_1)$. An argument similar to the proof for Theorem 10 can be used to construct a simulation relation.

So if $\rho_0, S_0, \mathscr{P}_0'$ has no trace such that there exists a k where $S_k \models (M \in e_2 \wedge M \notin e_1) \vee (M \in \textit{twice-}e_2)$ then it cannot simulate a trace from $\rho_0, S_0, \mathscr{P}_0$ such that $e_2(M)$ has fired but $e_1(M)$ has not, nor a trace where $e_2(M)$ has executed twice, and since $\rho_0, S_0, \mathscr{P}_0' \propto \rho_0, S_0, \mathscr{P}_0$ then there is no trace violating the injective agreement.

A.3 Correctness of the Analysis

DEFINITION 7 (ORDER RELATION $\preceq_L$) *The order relation $S_1 \preceq_L S_2$ between states S_1 and S_2 holds iff:*

(i) $\forall s_j \in L \,.\, S_1(s_j) = S_2(s_j)$;

(ii) $\forall \mathsf{p}(M_1,\ldots,M_k) \,.\, (\!|\mathsf{p}(M_1,\ldots,M_k)|\!)_{S_1} \in \mathscr{F}_{P_0} \Rightarrow (\!|\mathsf{p}(M_1,\ldots,M_k)|\!)_{S_2} \in \mathscr{F}_{P_0}$.

PROOF.Let M be a term and s be a set. If $s \in L$ then $\alpha'(s,M) = \alpha(s,M)$ by definition of $relax(\alpha, L)$, and $S'(s) = S(s)$ by property *(i)* of the order relation $\preceq_L$. Therefore if $\alpha'(s,M) = \alpha(s,M) = 1[]$ then $M \in S(s) = S(s')$ and if $\alpha'(s,M) = \alpha(s,M) = 0[]$ then $M \notin S(s) = S'(s)$.

If $\alpha'(s,M) = x$ then either $s \in L$ and $\alpha'(s,M) = \alpha(s,M)$, or $s \notin L$ and $x = x_{s,M}$. In the first case x is different from all other $\alpha'(s',M')$ were $M' \neq M \wedge s' \neq s$, by hypothesis if $s' \in L$ and by construction if $s \notin L$. In the second case $x_{s,M}$ is unique by construction.

LEMMA 8 (*restrict* PRESERVES THE SET-ABSTRACTION) *Let α be a set abstraction, ρ an environment, S a state and $A = restrict(\alpha, b)$. If $\rho, S \models b$ and α abstracts S, then there exists an $\alpha' \in A$ such that α' abstracts S.*

PROOF.The proof is done by induction on the depth of b.

Base cases:

Case $b = M \in s$.
If $A = \emptyset$ then it is the case that $\alpha(s,M) = 0$, and because α abstracts S under ρ we have $M \notin S(s)$, hence $\rho, S \not\models M \in s$, invalidating the hypothesis.

If $A = \{\alpha'\}$ then $\alpha'(s,M) = 1$ and $\rho, S \models M \in s$ implies $\rho(M) \in S(s)$; and for all s', M' such that $s' \neq s$ or $M' \neq M$ we have $\alpha'(s',M') = \alpha(s',M')$. Therefore α' satisfies the abstraction requirements.

Case $b = \neg M \in s$.
Similar to $b = M \in s$.

Inductive cases:

Case $b = b_1 \wedge b_2$.
By definition $A = \bigcup\{restrict(\alpha', b_2) \mid \alpha' \in restrict(\alpha, b_1)\}$. If $\rho, S \models b_1 \wedge b_2$ then also

$\rho, S \models b_1$ and $\rho, S \models b_2$. By inductive hypothesis there exists $\alpha_1 \in \mathit{restrict}(\alpha, b_1)$ that abstracts S under ρ. For such α_1 again by inductive hypothesis there exists $\alpha_2 \in \mathit{restrict}(\alpha_1, b_2)$ that abstracts S under ρ, and α_2 is in A by construction.

Case $b = b_1 \vee b_2$**.**
By definition $A = \mathit{restrict}(\alpha, b_1) \cup \mathit{restrict}(\alpha, b_2)$, and since $\rho, S \models b_1 \vee b_2$ then (a) $\rho, S \models b_1$ or (b) $\rho, S \models b_2$. If (a) holds, then by inductive hypothesis there exists an $\alpha' \in \mathit{restrict}(\alpha, b_1)$ that abstracts S under ρ, therefore $\alpha' \in A$. Similar is the case when (b) holds.

Case $b = \neg(b_1 \wedge b_2)$**.**
Then by definition $A = \mathit{restrict}(\alpha, \neg(b_1 \wedge b_2)) = \mathit{restrict}(\alpha, (\neg b_1) \vee (\neg b_2))$ and by De Morgan's laws $\rho, S \models \neg(b_1 \wedge b_2)$ iff $\rho, S \models (\neg b_1) \vee (\neg b_2)$. The case for $\vee$ can then be applied, using the inductive hypothesis on $\neg b_1$ and $\neg b_2$.

Case $b = \neg(b_1 \vee b_2)$**.**
Then by definition $A = \mathit{restrict}(\alpha, \neg(b_1 \vee b_2)) = \mathit{restrict}(\alpha, (\neg b_1) \wedge (\neg b_2))$ and by De Morgan's laws $\rho, S \models \neg(b_1 \vee b_2)$ iff $\rho, S \models (\neg b_1) \wedge (\neg b_2)$. The case for $\wedge$ can then be applied, using the inductive hypothesis on $\neg b_1$ and $\neg b_2$.

Case $b = \neg\neg b_1$**.**
By definition $A = \mathit{restrict}(\alpha, \neg\neg b_1) = \mathit{restrict}(\alpha, b_1)$. Since $\rho, S \models \neg\neg b_1$ iff $\rho, S \models b_1$ we obtain the result by inductive hypothesis on b_1.

LEMMA 9 (implies **PRESERVES** $S \preceq_L S'$) *Let S be a set-membership state, and $S' = S \cup \{(s_1, M_1), \ldots, (s_j, M_j)\} \setminus \{(s_{j+1}, M_{j+1}), \ldots, (s_n, M_n)\}$. If for all $M \in \{M_1, \ldots, M_n\}$ we have* $\mathsf{implies}(\langle\!|M|\!\rangle_S, \langle\!|M|\!\rangle_{S'}) \in \mathscr{F}_{P_0}$ *then for any set of locks L such that $\{s_1, \ldots, s_n\} \cap L = \emptyset$ we have $S \preceq_L S'$.*

PROOF.The generated clauses for *implies* produce all the transitions over contexts that appear as conclusions in the clauses generated by the translation. Therefore if an attacker can derive a fact from a set of known terms $M_1, \ldots, M_n$ in state S, and there is a transition from state S to state S', then by induction either (base case) M_i is transferred because it is a head of one of the clauses produced for the protocol, or (inductive case) it is constructed by an attacker rule from a set of sub-terms $M'_1, \ldots, M'_k$ that are transferred by hypothesis.

LEMMA 10 (TYPABILITY OF A**)** *Let A be an attacker process, then $\rho_0, \emptyset, \emptyset, S_0 \Vdash A$.*

PROOF.Let B be a process, ρ an environment, S a state, V a list of terms. We prove that if:

(i) $\rho(B)$ is a closed process, $\rho(V)$ is ground,

(ii) $S_0 \preceq_\emptyset S$, and

(iii) for every maximal subterm M of B closed under ρ, we have $\{\!|\rho(\mathsf{att}(M))|\!\}_S \in \mathscr{F}_{P_0}$,

$$\rho, V, \emptyset, S \Vdash B$$

Proof by induction over the depth of B.

Base case:

Case $B = 0$**.**
Holds trivially:

$$\frac{}{\rho, V, \emptyset, S \Vdash 0}$$

Inductive cases:

Case $B = B_1 \,|\, B_2$**.**
We need to prove:

$$\frac{\forall S' \text{ s.t. } S \preceq_\emptyset S' \ (\rho, V, \emptyset, S' \Vdash B_1 \wedge \rho, V, \emptyset, S' \Vdash B_2)}{\rho, V, \emptyset, S \Vdash B_1 \,|\, B_2}$$

Let $i \in \{1,2\}$, let S' be a state such that $S \preceq_\emptyset S'$, then: (i) $\rho(B)$ is a closed process by hypothesis, $fv(B) = fv(B_i)$, hence $\rho(B_i)$ is also closed, $\rho(V)$ is ground by hypothesis; (ii) $S_0 \preceq_\emptyset S \preceq_\emptyset S'$ hence $S_0 \preceq_\emptyset S'$, (iii) because $S \preceq_\emptyset S'$ and because, by inductive hypothesis, for every maximal subterm M of B closed under ρ we have $\{\!|\rho(\mathsf{att}(M))|\!\}_S \in \mathscr{F}_{P_0}$, then $\{\!|\mathsf{att}(M)|\!\}_{S'} \in \mathscr{F}_{P_0}$ (condition iv of $\preceq_\emptyset$).

We proved $\rho, V, \emptyset, S' \Vdash B_1$ and $\rho, V, \emptyset, S' \Vdash B_2$ for any successor state S'; hence we conclude: $\rho, V, \emptyset, S' \Vdash B$.

Case $B = !^l B_1$**.**
We need to prove:

$$\frac{\forall S' \text{ s.t. } S \preceq_\emptyset S' \ (\rho \circ \{{}^{l}/_{x_l}\}, (x_l :: V), \emptyset, S' \Vdash B_1)}{\rho, V, \emptyset, S \Vdash !^l B_1}$$

where $l \in \mathbb{N}$.

Let S' be a state such that $S \preceq_\emptyset S'$; then: (i) $fv(B) = fv(B_1)$ and since $\rho(B)$ is closed then $\rho \circ \{^l/_{x_l}\}$ is also closed, as l does not appear in B; $(\rho \circ \{^l/_{x_l}\})(x_l :: V)$ is ground because $\rho(V)$ is ground by hypothesis and $x_l\{^l/_{x_l}\}$ is ground by construction; (ii) $S_0 \preceq_\emptyset S \preceq_\emptyset S'$ hence $S_0 \preceq_\emptyset S'$, (iii) because $S \preceq_\emptyset S'$ and because, by inductive hypothesis, for every maximal subterm M of B closed under ρ we have $\{\!|\rho(\mathsf{att}(M))|\!\}_S \in \mathscr{F}_{P_0}$, then $\{\!|\rho(\mathsf{att}(M))|\!\}_{S'} \in \mathscr{F}_{P_0}$.

Since we proved $\rho, V, \emptyset, S' \Vdash B_1$ for any successor state S', we can conclude that $\rho, V, \emptyset, S \Vdash B$.

Case $B = \mathbf{in}(M, x : T); B_1$**.**
We need to prove:

$$\frac{\begin{array}{l}\forall S' \text{ s.t. } S \preceq_\emptyset S' \ \forall N \text{ s.t. } \Gamma \vdash N : T \ \{\!|\rho'(\mathsf{msg}(M,N'))|\!\}_{S'} \\ \in \mathscr{F}_{P_0} \Rightarrow \rho', (N' :: V), \emptyset, S' \Vdash B_1\{^{N'}/_x\}\end{array}}{\rho, V, \emptyset, S \Vdash \mathbf{in}(M, x : T); B_1}$$

where $\rho' = \rho \circ mgu(N', N)$, $N' = pt_x^{ri(V)}(T)$.

Let S' be a state such that $S \preceq_\emptyset S'$, let N be a term such that $\Gamma \vdash N : T$; then: (i) since N is ground, ρ' is by construction a grounding substitution for all variables that appear in N'; therefore because $fv(B_1) = fv(B) \cup fv(N')$ and since B is closed under ρ, then B_1 is closed under ρ' and $\rho'(V \cup \{N'\})$ is ground (ii) $S_0 \preceq_\emptyset S \preceq_\emptyset S'$ hence $S_0 \preceq_\emptyset S'$; and (iii) since for every maximal subterm M' of B closed under ρ we have $\{\!|\rho(\mathsf{att}(M'))|\!\}_S \in \mathscr{F}_{P_0}$ by hypothesis and $S \preceq_\emptyset S'$, therefore we have $\{\!|\rho(\mathsf{att}(M'))|\!\}_{S'} \in \mathscr{F}_{P_0}$. In particular $\{\!|\rho(\mathsf{att}(M))|\!\}_{S'} \in \mathscr{F}_{P_0}$ and given that $\mathsf{att}(x_c) \wedge \mathsf{msg}(x_c, x_m) \Rightarrow \mathsf{att}(x_m) \in \mathscr{C}_{P_0}$ and $\{\!|\rho(\mathsf{msg}(M,N))|\!\}_{S'} \in \mathscr{F}_{P_0}$, we can conclude $\{\!|\rho(\mathsf{att}(N))|\!\}_{S'} \in \mathscr{F}_{P_0}$ and since $\rho(N) = \rho'(N')$ then $\{\!|\rho'(\mathsf{att}(N'))|\!\}_{S'} \in \mathscr{F}_{P_0}$. For every maximal subterm M' of $B_1\{^{N'}/_x\}$ that is closed under ρ', one of the following holds: either M' is also a maximal subterm of B closed under ρ, and since $Dom(\rho) \subseteq Dom(\rho')$ we have $\{\!|\rho'(\mathsf{att}(M'))|\!\}_{S'} \in \mathscr{F}_{P_0}$; or $M' = N'$ and hence $\{\!|\rho'(\mathsf{att}(N'))|\!\}_{S'} \in \mathscr{F}_{P_0}$, or $M' = f(M_1, \ldots, M_n)$ and $N' = M_i$ for some i; for $1 \leq j \leq n$, $j \neq i$, M_i is a maximal subterm of B closed under ρ, therefore $\{\!|\rho(\mathsf{att}(M_j))|\!\}_S \in \mathscr{F}_{P_0}$ by hypothesis, then also $\{\!|\rho'(\mathsf{att}(M_j))|\!\}_{S'} \in \mathscr{F}_{P_0}$, and since $\mathsf{att}(x_1) \wedge \cdots \wedge \mathsf{att}(x_n) \Rightarrow \mathsf{att}(f(x_1, \ldots, x_n))$ then also $\{\!|\rho'(\mathsf{att}(f(M_1, \ldots, M_n)))|\!\}_{S'} \in \mathscr{F}_{P_0}$; therefore we can conclude that $\rho', V, \emptyset, S' \Vdash B_1\{^{N'}/_x\}$ therefore also $\rho, V, \emptyset, S \Vdash \mathbf{in}(M, x : T); B_1$ holds.

Case $B = \mathbf{out}(M, N); B_1$**.**
We need to prove:

$$\frac{\{\!|\rho(\mathsf{msg}(M,N))|\!\}_S \in \mathscr{F}_{P_0} \wedge \forall S' \text{ s.t. } S \preceq_\emptyset S' \ (\rho, V, \emptyset, S' \Vdash B_1)}{\rho, V, \emptyset, S \Vdash \mathbf{out}(M, N); B_1}$$

Let us prove the first condition of the rule. By hypothesis, M and N are bound terms in

B, therefore $\langle\!|\rho(\mathsf{att}(M))|\!\rangle_S \in \mathscr{F}_{P_0}$ and $\langle\!|\rho(\mathsf{att}(N))|\!\rangle_S \in \mathscr{F}_{P_0}$. Since $\mathsf{att}(x_c) \wedge \mathsf{att}(x_m) \Rightarrow \mathsf{msg}(x_c, x_m) \in \mathscr{C}_{P_0}$ we conclude that $\langle\!|\rho(\mathsf{msg}(M,N))|\!\rangle_S \in \mathscr{F}_{P_0}$.

To prove the second condition, let S' be a state such that $S \preceq_\emptyset S'$; then: (i) $\rho(B)$ is a closed process by hypothesis, and $fv(B_1) \subseteq fv(B)$, therefore $\rho(B_1)$ is also closed; $\rho(V)$ is ground by hypothesis; (ii) $S_0 \preceq_\emptyset S \preceq_\emptyset S'$ hence $S_0 \preceq_\emptyset S'$; (iii) any maximal subterm M' of B closed under ρ is also a maximal subterm of B_1 closed under ρ; by hypothesis then $\langle\!|\rho(\mathsf{att}(M'))|\!\rangle_S \in \mathscr{F}_{P_0}$ and since $S \preceq_\emptyset S'$ then $\langle\!|\rho(\mathsf{att}(M'))S'|\!\rangle_\in \mathscr{F}_{P_0}$ and therefore $\rho, V, \emptyset, S' \Vdash B_1$.

Since both conditions of the rule are satisfied, we can conclude $\rho, V, \emptyset, S \Vdash \mathbf{out}(M,N); B_1$.

Case $B = \mathbf{new}^l\ x : a; B_1$.
We need to prove:

$$\frac{\begin{array}{l}\langle\!|\mathsf{name}(a^l[V])|\!\rangle_S \in \mathscr{F}_{P_0} \wedge \\ \forall S' \text{ s.t. } S \preceq_\emptyset S'\ \rho, V, \emptyset, S' \Vdash B_1\{^{a^l[V]}/_x\}\end{array}}{\rho, V, \emptyset, S \Vdash \mathbf{new}^l\ x : a; B_1}$$

because x is restricted within the attacker process A.

By definition of the attacker rules, $\langle\!|\mathsf{name}(a^\top[])|\!\rangle_{S_0} \in \mathscr{F}_{P_0}$, and since $l \notin label(P_0)$, then $\langle\!|a^l[V]|\!\rangle_{S_0} = \langle\!|a^\top[]|\!\rangle_{S_0}$, hence $\langle\!|\mathsf{name}(a^l[V])|\!\rangle_{S_0} \in \mathscr{F}_{P_0}$. Since $S_0 \preceq_\emptyset S$ then $\langle\!|\mathsf{name}(a^l[V])|\!\rangle_S \in \mathscr{F}_{P_0}$. Similarly we conclude that also $\langle\!|\mathsf{att}(a^l[V])|\!\rangle_{S'} \in \mathscr{F}_{P_0}$.

Let S' be a state such that $S \preceq_\emptyset S'$; then: (i) because $fv(B_1\{^{a^l[V]}/_x\}) = (fv(B) \cup \{x\}) \setminus \{x\} = fv(B)$ and by hypothesis $\rho(B)$ is closed, then also $\rho(B_1\{^{a^l[V]}/_x\})$ is closed; $\rho(V)$ is ground by hypothesis; (ii) since $S_0 \preceq_\emptyset S \preceq_\emptyset S'$ then $S_0 \preceq_\emptyset S'$, and (iii) by hypothesis, for every maximal subterm M of B closed under ρ we have $\langle\!|\rho(\mathsf{att}(M))|\!\rangle_S \in \mathscr{F}_{P_0}$, and since $S \preceq_\emptyset S'$ we also have $\langle\!|\rho(\mathsf{att}(M))|\!\rangle_{S'} \in \mathscr{F}_{P_0}$; we also have $\langle\!|\rho(\mathsf{att}(a^l[V]))|\!\rangle_{S'} \in \mathscr{F}_{P_0}$; every maximal subterm M of B_1 closed by ρ is either a maximal subterm of B closed by ρ, or it is the newly created name $a^l[V]$, or it is an applied constructor over maximal ρ-closed subterms of B: similarly to the case for input, in either situation the clause $\langle\!|\rho(\mathsf{att}(M))|\!\rangle_{S'}$ holds; therefore we satisfy the judgement $\rho, V, \emptyset, S' \Vdash B_1\{^{a[]}/_x\}$.

Because all conditions of the rule are satisfied we conclude $V, \emptyset, S \Vdash \mathbf{new}^l\ x : a; B_1$.

Case $B = \mathbf{let}\ x = g(M_1, \ldots, M_n)\ \mathbf{in} B_1\ \mathbf{else}\ B_2$.
We need to prove:

$$\frac{\begin{array}{l}\forall S' \text{ s.t. } S \preceq_\emptyset S'\ (\forall M \text{ s.t. } g(M_1, \ldots, M_n) \rightarrow_\rho M \\ \rho, V, \emptyset, S' \Vdash B_1\{^M/_x\}) \wedge \rho, V, \emptyset, S' \Vdash B_2\end{array}}{\rho, V, \emptyset, S \Vdash \mathbf{let}\ x = g(M_1, \ldots, M_n)\ \mathbf{in} B_1\ \mathbf{else}\ B_2}$$

Let S' be a state such that $S \preceq_\emptyset S'$.

We now prove $\rho, V, \emptyset, S' \Vdash B_1\{^M/_x\}$: (i) because $\rho(B)$ is closed, and because $fv(M) \subseteq fv(M_1, \ldots, M_n)$, then $fv(B_1\{^M/_x\}) = (fv(B) \cup \{x\}) \setminus \{x\} \cup fv(M) = fv(B)$, then also $\rho(B_1)$ is closed; (ii) since $S_0 \preceq_\emptyset S \preceq_\emptyset S'$ then $S_0 \preceq_\emptyset S'$, and (iii) by hypothesis, for every maximal subterm N of B closed under ρ we have $\langle\!|\rho(\mathsf{att}(N))|\!\rangle_S \in \mathscr{F}_{P_0}$, and since $S \preceq_\emptyset S'$ we also have $\langle\!|\rho(\mathsf{att}(N))|\!\rangle_{S'} \in \mathscr{F}_{P_0}$; in particular the property holds for $M_1, \ldots, M_n$ as they are bound in B; since for all destructor definitions:

$$\textbf{reduc } \forall \vec{x} : \vec{T} \,.\, g(M'_1, \ldots, M'_n) \rightarrow M'$$

$\mathscr{C}_{P_0}$ includes the rule:

$$\mathsf{att}(M'_1) \wedge \cdots \wedge \mathsf{att}(M'_n) \Rightarrow \mathsf{att}(M')$$

and because the rewrite rule succeeds producing M, then $\langle\!|\rho(\mathsf{att}(M_1))|\!\rangle_{S'} \in \mathscr{F}_{P_0}$ and $\langle\!|\rho(\mathsf{att}(M_n))|\!\rangle_{S'} \in \mathscr{F}_{P_0}$ and hence $\langle\!|\rho(\mathsf{att}(M))|\!\rangle_{S'} \in \mathscr{F}_{P_0}$. Hence the conditions (i–iii) are satisfied and we conclude $\rho, V, \emptyset, S' \Vdash B_1\{^M/_x\}$.

We now prove $\rho, V, \emptyset, S' \Vdash B_2$: (i) because σ is a grounding substitution for B, and since $fv(B) = fv(B_2)$ then it is also grounding for B_2; (ii) since $S_0 \preceq_\emptyset S \preceq_\emptyset S'$ then $S_0 \preceq_\emptyset S'$, and (iii) by hypothesis, for every maximal subterm M of B that is closed under ρ we have $\langle\!|\rho(\mathsf{att}(M))|\!\rangle_S \in \mathscr{F}_{P_0}$, and since $S \preceq_\emptyset S'$ we also have $\langle\!|\rho(\mathsf{att}(M))|\!\rangle_{S'} \in \mathscr{F}_{P_0}$; hence we conclude $\rho, V, \emptyset, S' \Vdash B_2$.

Since both hypotheses of the rule are satisfied, we conclude that $\rho, V, \emptyset, S \Vdash \textbf{let } x = g(M_1, \ldots, M_n) \textbf{ in } B_1 \textbf{ else } B_2$.

In particular, we have that (i) $fv(A) = x_{ch}$, hence $\rho_0(A)$ is closed; (ii) $S_0 \preceq_\emptyset S_0$ by reflexivity; and (iii) the only maximal subterm of A that is bound by ρ_0 is x_{ch}, and by construction of the translation $\langle\!|\rho(\mathsf{att}(x_{ch}))|\!\rangle_{S_0} \in \mathscr{F}_{P_0}$. Hence the attacker process types.

LEMMA 11 (TYPABILITY OF P_0) $\rho_0, \emptyset, \emptyset, S_0 \Vdash P_0$.

PROOF.Let Q be a process. We prove that, given a list of terms V, a set of locks L, a state S, a set-abstraction α, an environment ρ; if:

(i) $\rho(Q)$ is a closed process, $\rho(V)$ and $\rho(H)$ are ground,

(ii) α abstracts S,

(iii) $\mathscr{C}_{P_0} \supseteq [\![Q]\!]HVL\alpha$,

(iv) for every predicate p in H, we have that $\langle\!|\rho(p)|\!\rangle_S \in \mathscr{F}_{P_0}$

Then $\rho, V, L, S \Vdash Q$.

The proof is carried by induction on the size of the process Q.

Base case:

Case $Q = 0$**.**
Trivially holds:

$$\frac{}{\rho, V, L, S \Vdash 0}$$

Inductive cases:

Case $Q = Q_1 \,|\, Q_2$**.**
We need to prove:

$$\frac{\forall S' \text{ s.t. } S \preceq_\emptyset S' \ (\rho, V, \emptyset, S' \Vdash Q_1 \wedge \rho, V, \emptyset, S' \Vdash Q_2)}{\rho, V, \emptyset, S \Vdash Q_1 \,|\, Q_2}$$

Let $\alpha' = relax(\alpha, \emptyset)$, let S' be any state such that $S \preceq_\emptyset S'$.

Let $i \in \{1,2\}$, then: (i) $fv(Q_i) \subseteq fv(Q)$ hence $\rho(Q_i)$ is closed, and $\rho(V)$ and $\rho(H)$ are ground by hypothesis, (ii) α' abstracts S' under ρ by Lemma 3, (iii) $\mathscr{C}_{P_0} \supseteq [\![Q_1 \,|\, Q_2]\!] HV\emptyset\alpha \supseteq [\![Q_i]\!] HV\emptyset\alpha'$ by construction, (iv) by hypothesis, for every predicate p in H, we have that $\langle\!|\rho(p)|\!\rangle_S \in \mathscr{F}_{P_0}$ and $S \preceq_\emptyset S'$, then $\langle\!|\rho(p)|\!\rangle_{S'} \in \mathscr{F}_{P_0}$, we finally conclude that $\rho, V, \emptyset, S' \Vdash Q_i$.

Therefore we can conclude that also $\rho, V, \emptyset, S \Vdash Q_1 \,|\, Q_2$.

Case $Q = !^l Q_1$**.**
We need to prove:

$$\frac{\forall S' \text{ s.t. } S \preceq_\emptyset S' \ (\rho \circ \{{}^l/_{x_l}\}, (x_l :: V), \emptyset, S' \Vdash Q_1)}{\rho, V, \emptyset, S \Vdash !^l Q_1} \ l \in \mathbb{N}$$

Let $\alpha' = relax(\alpha, \emptyset)$, let S' be any state such that $S \preceq_\emptyset S'$ and $\rho' = \rho \circ \{{}^l/_{x_l}\}$.

Because (i) $fv(Q_1) = fv(Q)$ hence $\rho'(Q_1)$ is closed, and $\rho'(x_l :: V)$ is ground since x_l is ground by the applied substitution $\{{}^l/_{x_l}\}$ and $\rho(V)$ is ground by inductive hypothesis,

$\rho(H)$ is ground by hypothesis hence also $\rho'(H)$ is ground, (ii) α' abstracts S' under ρ by Lemma 3, (iii) $\mathscr{C}_{P_0} \supseteq [\![!^l\, Q_1]\!]HV\emptyset\alpha = [\![Q_1]\!]H(x_l :: V)\emptyset\alpha'$, and (iv) by hypothesis, for every predicate p in H, we have that $\{\!|\rho(p)|\!\}_S \in \mathscr{F}_{P_0}$ and $S \preceq_\emptyset S'$, then $\{\!|\rho(p)|\!\}_{S'} \in \mathscr{F}_{P_0}$, we finally conclude that $\rho \circ \{^l/_{x_l}\}, (x_l :: V), \emptyset, S' \Vdash Q_1$.

Therefore we can conclude that also $\rho, V, \emptyset, S \Vdash !^l\, Q_1$.

Case $Q = \mathbf{in}(M, x : T); Q_1$**.**
We need to prove:

$$\frac{\begin{array}{l}\forall S' \text{ s.t. } S \preceq_L S'\ \forall N \text{ s.t. } \Gamma \vdash N : T\ \{\!|\rho(\mathsf{msg}(M,N))|\!\}_{S'} \\ \in \mathscr{F}_{P_0} \Rightarrow \rho', (N' :: V), L, S' \Vdash Q_1\{^{N'}/_x\}\end{array}}{\rho, V, L, S \Vdash \mathbf{in}(M, x : T); Q_1}$$

where $N' = pt_x^{ri(V)}(T)$.

Let $\alpha' = relax(\alpha, L)$, let S' be any state such that $S \preceq_L S'$, let $\rho' = \rho \circ mgu(N', N)$.

Because (i) $fv(Q_1\{^{N'}/_x\}) = (fv(Q) \cup \{x\}) \setminus \{x\} \cup fv(N') \subseteq Dom(\rho')$, and $\rho'(N' :: V)$ is ground because by construction $\rho'(N')$ is ground, and $\rho(V)$ is ground by hypothesis, and $\rho'(H \wedge \mathsf{msg}(M, N'))$ is also ground, because $\rho'(N')$ is ground by construction, and $\rho(M)$ is ground by hypothesis, since $\rho(Q)$ is closed; (ii) α' abstracts S' by Lemma 3, (iii) $\mathscr{C}_{P_0} \supseteq [\![P\{^{N'}/_x\}]\!](H \wedge \mathsf{msg}(M, N'))(N' :: V)L\alpha'$ and (iv) by hypothesis, for every predicate p in H, we have that $\{\!|\rho(p)|\!\}_S \in \mathscr{F}_{P_0}$ and $S \preceq_L S'$, then $\{\!|\rho(p)|\!\}_{S'} \in \mathscr{F}_{P_0}$, and by hypothesis on the rule we require $\{\!|\rho(\mathsf{msg}(M, N))|\!\}_{S'} \in \mathscr{F}_{P_0}$, therefore we conclude that $\rho', (N' :: V), L, S' \Vdash Q_1\{^{N'}/_x\}$.

Therefore we can conclude that also $\rho, V, L, S \Vdash \mathbf{in}(M, x : T); Q_1$.

Case $Q = \mathbf{out}(M, N); Q_1$**.**
We need to prove:

$$\frac{\begin{array}{l}\{\!|\rho(\mathsf{msg}(M,N))|\!\}_S \in \mathscr{F}_{P_0} \wedge \\ \forall S' \text{ s.t. } S \preceq_L S'\ (\rho, V, L, S' \Vdash Q_1)\end{array}}{\rho, V, L, S \Vdash \mathbf{out}(M, N); Q_1}$$

Let $\alpha' = relax(\alpha, L)$, and let S' be any state such that $S \preceq_L S'$.

Because (i) $fv(Q_1) = fv(Q)$ and $\rho(Q)$ is closed, hence $\rho(Q_1)$ is closed, by inductive hypothesis $\rho(V)$ and $\rho(H)$ are ground, (ii) α' abstracts S' by Lemma 3, (iii) $\mathscr{C}_{P_0} \supseteq [\![Q_1]\!]HVL\alpha'$ and (iv) by hypothesis, for every predicate p in H, we have that $\{\!|\rho(p)|\!\}_S \in \mathscr{F}_{P_0}$ and $S \preceq_L S'$, then $\{\!|\rho(p)|\!\}_{S'} \in \mathscr{F}_{P_0}$, hence we conclude that $\rho, V, L, S' \Vdash Q_1$.

Since by construction $\langle\!|H \Rightarrow \mathsf{msg}(M,N)|\!\rangle_\alpha \in \mathscr{C}_{P_0}$, and by hypothesis α abstracts S and for every predicate p in H holds $\langle\!|\rho(p)|\!\rangle_S \in \mathscr{F}_{P_0}$, we can conclude that $\langle\!|\rho(\mathsf{msg}(M,N))|\!\rangle_S \in \mathscr{F}_{P_0}$.

Therefore all hypotheses of the rule for output are satisfied, so we can conclude that $\rho,V,L,S \Vdash \mathbf{out}(M,N); Q_1$.

Case $Q = \mathbf{new}^l\ x : a; Q_1$.
We need to prove:

$$\frac{\begin{array}{l}\langle\!|\rho(\mathsf{name}(a^l[V]))|\!\rangle_S \in \mathscr{F}_{P_0} \wedge \\ \forall S' \text{ s.t. } S \preceq_L S'\ \rho,V,L,S' \Vdash Q_1\{^{a^l[V]}/_x\}\end{array}}{\rho,V,L,S \Vdash \mathbf{new}^l\ x : a; Q_1}$$

Since by construction $\langle\!|H \Rightarrow \mathsf{name}(a^l[V])|\!\rangle_\alpha \in \mathscr{C}_{P_0}$, and by hypothesis α abstracts S and for every predicate p in H holds $\langle\!|\rho(p)|\!\rangle_S \in \mathscr{F}_{P_0}$, we can conclude that $\langle\!|\rho(\mathsf{name}(a^l[V]))|\!\rangle_S \in \mathscr{F}_{P_0}$.

Let $\alpha' = \mathit{relax}(\alpha,L)$, let S' be any state such that $S \preceq_L S'$.

Because (i) $\mathit{fv}(Q_1\{^{a^l[V]}/_x\}) = (\mathit{fv}(Q) \cup \{x\}) \setminus \{x\} \cup \mathit{fv}(V)$ and since $\rho(Q)$ and $\rho(V)$ are closed by inductive hypothesis, also $\rho(Q_1\{^{a^l[V]}/_x\})$ is closed; $\rho(H)$ and $\rho(V)$ are closed by inductive hypothesis; (ii) α' abstracts S' by Lemma 3, (iii) $\mathscr{C}_{P_0} \supseteq [\![Q_1\{^{a^l[V]}/_x\}]\!](H \wedge \mathsf{name}(a^l[V]))VL\alpha'$ and (iv) by hypothesis, for every predicate p in H, we have that $\langle\!|\rho(p)|\!\rangle_S \in \mathscr{F}_{P_0}$ and $S \preceq_L S'$, then $\langle\!|\rho(p)|\!\rangle_{S'} \in \mathscr{F}_{P_0}$, and since we proved $\langle\!|\rho(\mathsf{name}(a^l[V]))|\!\rangle_S \in \mathscr{F}_{P_0}$ and $S \preceq_L S'$ then $\langle\!|\rho(\mathsf{name}(a^l[V]))|\!\rangle_{S'} \in \mathscr{F}_{P_0}$, therefore we conclude that $\rho,V,L,S' \Vdash Q_1\{^{a^l[V]}/_x\}$.

Therefore all hypotheses of the rule for restriction are satisfied, so we can conclude that $\rho,V,L,S \Vdash \mathbf{new}^l\ x : a; Q_1$.

Case $Q = \mathbf{let}\ x = g(M_1,\ldots,M_n)\ \mathbf{in}\ Q_1\ \mathbf{else}\ Q_2$.
We need to prove:

$$\frac{\begin{array}{l}\forall S' \text{ s.t. } S \preceq_L S'\ (\forall M \text{ s.t. } g(M_1,\ldots,M_n) \rightarrow_\rho M \\ \rho,V,L,S' \Vdash Q_1\{^M/_x\}) \wedge \rho,V,L,S' \Vdash Q_2\end{array}}{\rho,V,L,S \Vdash \mathbf{let}\ x = g(M_1,\ldots,M_n)\ \mathbf{in}\ Q_1\ \mathbf{else}\ Q_2}$$

Let $\alpha' = \mathit{relax}(\alpha,L)$, let S' be any state such that $S \preceq_L S'$. Assume $g(M_1,\ldots,M_n) \rightarrow_\rho M$.

As for the Q_1 branch, the following clauses will be generated:

$$\{[\![\sigma(P_1)]\!]\sigma(H)\sigma(V)L\alpha'' \mid \mathbf{reduc} \forall \vec{x'} : \vec{T'}$$

$g(M'_1,\ldots,M'_n) \to M'$ is in the scope of **let**, σ is an m.g.u.
that satisfies $M_1 \stackrel{\circ}{=} M'_1 \wedge \cdots \wedge M_n \stackrel{\circ}{=} M'_n \wedge x \stackrel{\circ}{=} M', \theta$ is
an m.g.u. that satisfies, $\forall s, N_1, N_2$, $\sigma(N_1) = \sigma(N_2) \Rightarrow$
$\alpha'(s,N_1) \stackrel{\circ}{=} \alpha'(s,N_2)$ and $\alpha''(s,\sigma(N_1)) = \theta(\alpha'(s,N_1))\}$

Because (i) $fv(Q_1\{^M/_x\}) = (fv(Q) \cup \{x\}) \setminus x \cup fv(M)$ and we require that $fv(M) \subseteq fv(M_1,\ldots,M_n)$, then $fv(Q_1\{^M/_x\}) \subseteq fv(Q) \cup fv(M_1,\ldots,M_n)$; since $\rho(Q)$ is closed then also $\rho(M_1),\ldots,\rho(M_n)$ are closed, hence we conclude that $\rho(Q_1\{^M/_x\})$ is also closed; $\rho(H)$ and $\rho(V)$ are ground by hypothesis; (ii) α' abstracts S' under ρ by Lemma 3, (iii) $\mathscr{C}_{P_0} \supseteq [\![\sigma(Q_1)]\!]\sigma(H)\sigma(V)L\alpha'$ and (iv) by hypothesis, for every predicate p in H, we have that $\{\!|\rho(p)|\!\}_S \in \mathscr{F}_{P_0}$ and $S \preceq_L S'$, then $\{\!|\rho(p)|\!\}_{S'} \in \mathscr{F}_{P_0}$, therefore we conclude that $\rho,V,L,S' \Vdash Q_1\{^M/_x\}$.

As for the Q_2 branch: (i) $\rho(Q)$ is closed, hence also $\rho(Q_2)$; $\rho(V)$ and $\rho(H)$ are ground by hypothesis, (ii) α' abstracts S' under ρ by Lemma 3, (iii) $\mathscr{C}_{P_0} \supseteq [\![Q_2]\!]HVL\alpha'$ by definition of the translation, and (iv) by hypothesis, for every predicate p in H, we have that $\{\!|\rho(p)|\!\}_S \in \mathscr{F}_{P_0}$ and $S \preceq_L S'$, then $\{\!|\rho(p)|\!\}_{S'} \in \mathscr{F}_{P_0}$, we conclude that $\rho,V,L,S' \Vdash Q_1$.

Therefore all hypotheses of the rule for let are satisfied, so we can conclude that $\rho,V,L,S \Vdash$ **let** $x = g(M_1,\ldots,M_n)$ **in** Q_1 **else** Q_2.

Case $Q =$ **if** b **then** Q_1 **else** Q_2**.**
We need to prove:

$$\frac{\begin{array}{l}\forall S' \text{ s.t. } S \preceq_L S',\ (\rho,S' \models b \Rightarrow \rho,V,L,S' \Vdash Q_1) \wedge \\ (\rho,S' \models \neg b \Rightarrow \rho,V,L,S' \Vdash Q_2)\end{array}}{\rho,V,L,S \Vdash \textbf{if } b \textbf{ then } Q_1 \textbf{ else } Q_2}$$

Let $\alpha' \in restrict(relax(\alpha,L),b)$, let S' be any state such that $S \preceq_L S'$. Assume $\rho,S' \models b$.

Because (i) $\rho(Q)$ is closed, hence also $\rho(Q_1)$; $\rho(V)$ and $\rho(H)$ are ground by hypothesis, (ii) α' abstracts S' under ρ and is guaranteed to exist by Lemma 3 and Lemma 8 since $\rho,S \models b$, (iii) $\mathscr{C}_{P_0} \supseteq [\![Q_1]\!]HVL\alpha'$ by definition of the translation, and (iv) by hypothesis, for every predicate p in H, we have that $\{\!|\rho(p)|\!\}_S \in \mathscr{F}_{P_0}$ and $S \preceq_L S'$, then $\{\!|\rho(p)|\!\}_{S'} \in \mathscr{F}_{P_0}$, we conclude that $\rho,V,L,S' \Vdash Q_1$.

A similar argument, taking $\alpha' \in restrict(relax(\alpha,L),\neg b)$ and assuming $\rho,S' \models \neg b$, proves $\rho,V,L,S' \Vdash Q_2$.

Therefore all hypotheses of the rule for let are satisfied, so we can conclude that $\rho,V,L,S \Vdash$ **if** b **then** Q_1 **else** Q_2.

Case $Q = \mathbf{lock}(s); Q_1$**.**
We need to prove:

$$\frac{\forall S' \text{ s.t. } S \preceq_L S' \ \rho, V, (L \cup \{s\}), S' \Vdash Q_1}{\rho, V, L, S \Vdash \mathbf{lock}(s); Q_1}$$

Let $\alpha' = \mathit{relax}(\alpha, L)$, let S' be any state such that $S \preceq_L S'$.

Because (i) $\rho(Q)$ is closed, hence also $\rho(Q_1)$; $\rho(V)$ and $\rho(H)$ are ground by hypothesis, (ii) α' abstracts S' under ρ by Lemma 3, (iii) $\mathscr{C}_{P_0} \supseteq [\![Q_1]\!] HV(L \cup \{s\})\alpha'$ by definition of the translation, and (iv) by hypothesis, for every predicate p in H, we have that $\{\!|\rho(p)|\!\}_S \in \mathscr{F}_{P_0}$ and $S \preceq_L S'$, then $\{\!|\rho(p)|\!\}_{S'} \in \mathscr{F}_{P_0}$, we conclude that $\rho, V, (L \cup \{s\}), S' \Vdash Q_1$.

Therefore all hypotheses of the rule for lock are satisfied, so we can conclude that $\rho, V, L, S \Vdash \mathbf{lock}(s); Q_1$.

Case $Q = \mathbf{unlock}(s); Q_1$**.**
We need to prove:

$$\frac{\forall S' \text{ s.t. } S \preceq_L S' \ \rho, V, (L \setminus \{s\}), S' \Vdash Q_1}{\rho, V, L, S \Vdash \mathbf{unlock}(s); Q_1}$$

Let $\alpha' = \mathit{relax}(\alpha, L)$, let S' be any state such that $S \preceq_L S'$.

Because (i) $\rho(Q)$ is closed, hence also $\rho(Q_1)$; $\rho(V)$ and $\rho(H)$ are ground by hypothesis, (ii) α' abstracts S' under ρ by Lemma 3, (iii) $\mathscr{C}_{P_0} \supseteq [\![Q_1]\!] HV(L \setminus \{s\})\alpha'$ by definition of the translation, and (iv) by hypothesis, for every predicate p in H, we have that $\{\!|\rho(p)|\!\}_S \in \mathscr{F}_{P_0}$ and $S \preceq_L S'$, then $\{\!|\rho(p)|\!\}_{S'} \in \mathscr{F}_{P_0}$, we conclude that $\rho, V, (L \setminus \{s\}), S' \Vdash Q_1$.

Therefore all hypotheses of the rule for unlock are satisfied, so we can conclude that $\rho, V, L, S \Vdash \mathbf{unlock}(s); Q_1$.

Case $Q = \mathbf{set}(b^+); Q_1$**.**
We need to prove:

$$\frac{\begin{array}{l}\forall S' \text{ s.t. } S \preceq_L S' \ (\forall M \in \mathit{fv}(b^+) \cup \mathit{fn}(b^+) \\ \quad \mathsf{implies}(\{\!|\rho(M)|\!\}_{S'}, \{\!|\rho(M)|\!\}_{S''}) \in \mathscr{F}_{P_0}) \wedge \\ \quad (\forall S''' \text{ s.t. } S'' \preceq_L S''', \ \rho, V, L, S''' \Vdash Q_1)\end{array}}{\rho, V, L, S \Vdash \mathbf{set}(b^+); Q_1}$$

where $S'' = \mathit{update}(S', \rho(b^+))$.

Let $\alpha' = relax(\alpha, L)$, let S' be any state such that $S \preceq_L S'$.

Because by hypothesis for all p in H we have that $\langle\!|p|\!\rangle_S \in \mathscr{F}_{P_0}$, and $S \preceq_L S'$ then $\langle\!|p|\!\rangle_{S'} \in \mathscr{F}_{P_0}$.

Let $\alpha'' = update(\alpha', b^+) = \alpha' \cup \{(s_1, M_1), \ldots, (s_j, M_j)\} \setminus \{(s_{j+1}, M_{j+1}), \ldots, (s_n, M_n)\}$. Because also $\mathscr{C}_{P_0} \supseteq \{\langle\!|H|\!\rangle_{\alpha'} \Rightarrow \mathsf{implies}(\langle\!|M|\!\rangle_{\alpha'}, \langle\!|M|\!\rangle_{\alpha''}) \mid M \in \{M_1, \ldots, M_n\}\}$ by the translation, and α' abstracts S' under ρ by Lemma 3, we obtain that $\forall\, M \in fv(b^+) \cup fn(b^+)$ $\mathsf{implies}(\langle\!|\rho(M)|\!\rangle_{S'}, \langle\!|\rho(M)|\!\rangle_{S''}) \in \mathscr{F}_{P_0}$ and hence the first condition is satisfied.

We are left to prove for every state S''' such that $S'' \preceq_L S'''$ we have $\rho, V, L, S''' \Vdash Q_1$.

We know that: (i) $\rho(Q)$ is closed, hence also $\rho(Q_1)$; $\rho(V)$ and $\rho(H)$ are ground by hypothesis, (ii) α' abstracts S' under ρ by Lemma 3, hence $\alpha'' = update(\alpha, b^+)$ abstracts S'' under ρ by Lemma 9, therefore also $\alpha''' = relax(\alpha'', L)$ abstracts S''' under ρ again by Lemma 3; (iii) $\mathscr{C}_{P_0} \supseteq [\![Q_1]\!]HVL(\alpha' \cup \{(s, M)\})$ by definition of the translation, and (iv) for every predicate p in H we proved that $\langle\!|\rho(p)|\!\rangle_{S'} \in \mathscr{F}_{P_0}$ and that $\mathsf{implies}(\langle\!|\rho(M)|\!\rangle_{S'}, \langle\!|\rho(M)|\!\rangle_{S''}) \in \mathscr{F}_{P_0}$; by Lemma 9 we have that also $\langle\!|\rho(p)|\!\rangle_{S''} \in \mathscr{F}_{P_0}$; finally because $S'' \preceq_L S'''$ we also obtain that $\langle\!|\rho(p)|\!\rangle_{S'''} \in \mathscr{F}_{P_0}$.

Hence conditions (i-iv) are satisfied so $\rho, V, L, S''' \Vdash Q_1$.

In particular, (i) ρ_0 closes P_0 by construction, $\rho_0(\emptyset)$ is trivially ground, (ii) α_0 abstracts S_0 under ρ_0 by construction, (iii) $\mathscr{C}_{P_0} \supseteq [\![P_0]\!]\emptyset\emptyset\emptyset\alpha_0$ by definition of the translation, (iv) holds vacuously. Therefore the conditions (i–iv) are satisfied and hence $\rho_0, \emptyset, \emptyset, S_0 \Vdash P_0$.

THEOREM 12 (SUBJECT REDUCTION) *If $\rho, S, \mathscr{P} \to \rho', S', \mathscr{P}'$ and for all $(P, L, V) \in \mathscr{P}$ we have $\rho, V, L, S \Vdash P$ then for all $(P', L', V') \in \mathscr{P}'$ we have $\rho', V', L', S' \Vdash P'$.*

PROOF. We prove subject reduction with a case-by-case analysis on the semantic steps.

Case NIL.
$\rho, S, \mathscr{P} \uplus \{(0, L, V)\} \to \rho, S, \mathscr{P}$

By hypothesis for all $(P, L, V) \in \mathscr{P}$ we have $\rho, V, L, S \Vdash P$, and this trivially holds also after the transition.

Case COM.
$\rho, S, \mathscr{P} \uplus \{(\mathbf{in}(M, x{:}T); P_1, L_1, V_1), (\mathbf{out}(M, N); P_2, L_2, V_2)\}$
$\to \rho', S, \mathscr{P} \uplus \{(P_1\{^{N'}/_x\}, L_1, N' :: V_1), (P_2, L_2, V_2)\}$ where $\Gamma \vdash N : T$ and $\rho' = \rho \circ mgu(N', N)$ and $N' = pt_x^{ri(V)}(T)$.

All processes in $\mathscr{P}$ type after the transition: in particular, they are not influenced by the variables introduced in the domain of ρ, which are enforced to be syntactically different (Lemma 2). By hypothesis:

$$\frac{\begin{array}{l}\forall S' \text{ s.t. } S \preceq_{L_1} S' \; \forall N'' \text{ s.t. } \Gamma \Vdash N'' : T \\ \{\!|\rho(\mathsf{msg}(M,N''))|\!\}_{S'} \in \mathscr{F}_{P_0} \Rightarrow \\ (\rho \circ mgu(N''',N'')), (N' :: V_1), L_1, S' \Vdash P_1\{^{N'''}/_x\}\end{array}}{\rho, V_1, L_1, S \Vdash \mathbf{in}(M, x:T); P_1}$$

and

$$\frac{\begin{array}{l}\{\!|\rho(\mathsf{msg}(M,N))|\!\}_S \in \mathscr{F}_{P_0} \wedge \\ \forall S' \text{ s.t. } S \preceq_{L_2} S' \; (\rho, V_2, L_2, S' \Vdash P_2)\end{array}}{\rho, V_2, L_2, S \Vdash \mathbf{out}(M,N); P_2}$$

Therefore $\{\!|\rho(\mathsf{msg}(M,N))|\!\}_S \in \mathscr{F}_{P_0}$ according to the deduction rule for output. Since also $S \preceq_{L_1} S$ and $\Gamma \vdash N : T$ by hypothesis, by the rule for input we have that $(\rho \circ mgu(N',N)), (N' :: V_1), L_1, S \Vdash P_1\{^{N'}/_x\}$. Finally, by the rule for output and because $S \preceq_{L_2} S$, and by the extension lemma (Lemma 2) on ρ', we have $\rho', V_2, L_2, S \Vdash P_2$.

Case PAR.
$\rho, S, \mathscr{P} \uplus \{(P_1 \,|\, P_2, \emptyset, V)\} \to \rho, S, \mathscr{P} \uplus \{(P_1, \emptyset, V), (P_2, \emptyset, V)\}$

All processes in $\mathscr{P}$ trivially type after the transition. By hypothesis:

$$\frac{\forall S' \text{ s.t. } S \preceq_\emptyset S' \; (\rho, V, \emptyset, S' \Vdash P_1 \;\wedge\; \rho, V, \emptyset, S' \Vdash P_2)}{\rho, V, \emptyset, S \Vdash P_1 \,|\, P_2}$$

In particular because $S \preceq_\emptyset S$ we have that $\rho, V, \emptyset, S \Vdash P_1$ and $\rho, V, \emptyset, S \Vdash P_2$.

Case REPL.
$\rho, S, \mathscr{P} \uplus \{(!^k P, \emptyset, V)\} \to$
$\rho \circ \{^k/_{x_k}\}, S, \mathscr{P} \uplus \{(P, \emptyset, x_k :: V), (!^{k+1} P, \emptyset, V)\}$

All processes in $\mathscr{P}$ trivially type after the transition, as x_k does not appear in any of them by construction. By hypothesis:

$$\frac{\forall S' \text{ s.t. } S \preceq_\emptyset S' \; (\rho \circ \{^l/_{x_l}\}, (x_l :: V), \emptyset, S' \Vdash P)}{\rho, V, \emptyset, S \Vdash !^k P} \; l \in \mathbb{N}$$

with $l = k$. The rule can be applied with $l = k+1$ and hence also $\rho \circ \{^k/_{x_k}\}, V, \emptyset, S \Vdash !^{k+1} P$, since x_k does not appear in the process $!^{k_1} P$. In particular $S \preceq_\emptyset S$ and therefore by the extension lemma (Lemma 2) $\rho \circ \{^k/_{x_k}\}(V \cup \{i\}), \emptyset, S \Vdash P$.

Case NEW.
$\rho, S, \mathscr{P} \uplus \{(\mathbf{new}^l \; x : a; P, L, V)\} \to$
$\rho, S, \mathscr{P} \uplus \{(P\{^{a^l[V]}/_x\}, L, V)\}$

All processes in $\mathscr{P}$ trivially type after the transition.

By hypothesis:

$$\frac{\begin{array}{l} x \in bv(P_0) \Rightarrow (\{\!|\rho(\mathsf{name}(a[V]))|\!\}_S \in \mathscr{F}_{P_0} \wedge \\ \forall S' \text{ s.t. } S \preceq_L S'\ \rho, V, L, S' \Vdash P\{^{a^l[V]}/_x\}) \end{array}}{\rho, V, L, S \Vdash \mathbf{new}^l\ x : a; P}$$

In particular $S \preceq_L S$ and hence the judgement $\rho, V, L, S \Vdash P\{^{a^l[V]}/_x\}$ holds.

Case LET-1.
$\rho, S, \mathscr{P} \uplus \{(\mathbf{let}\ x = g(M_1, \ldots, M_n)\ \mathbf{in} P_1\ \mathbf{else}\ P_2, L, V)\} \rightarrow$
$\rho, S, \mathscr{P} \uplus \{(P_1\{^M/_x\}, L, V)\}$ and $g(M_1, \ldots, M_n) \rightarrow_\rho M$

All processes in $\mathscr{P}$ trivially type after the transition.

By hypothesis:

$$\frac{\begin{array}{l} \forall S' \text{ s.t. } S \preceq_L S'\ (\forall M \text{ s.t. } g(M_1, \ldots, M_n) \rightarrow_\rho M \\ \rho, V, L, S' \Vdash P_1\{^M/_x\}) \wedge \rho, V, L, S' \Vdash P_2 \end{array}}{\rho, V, L, S \Vdash \mathbf{let}\ x = g(M_1, \ldots, Q_n)\ \mathbf{in} P_1\ \mathbf{else}\ P_2}$$

Since $S \preceq_L S$ we obtain $\rho, V, L, S \Vdash P_1\{^M/_x\}$.

Case LET-2.
$\rho, S, \mathscr{P} \uplus \{(\mathbf{let}\ x = g(M_1, \ldots, M_n)\ \mathbf{in} P_1\ \mathbf{else}\ P_2, L, V)\} \rightarrow$
$\rho, S, \mathscr{P} \uplus \{(P_2, L, V)\}$ and $g(M_1, \ldots, M_n) \nrightarrow_\rho$

Similarly to the case LET-1, all processes in $\mathscr{P}$ type after the transition, and $\rho, V, L, S \Vdash P_2$.

Case IF-1.
$\rho, S, \mathscr{P} \uplus \{(\mathbf{if}\ b\ \mathbf{then} P_1\ \mathbf{else}\ P_2, L, V)\} \rightarrow$
$\rho, S, \mathscr{P} \uplus \{(P_1, L, V)\}$ and $S \models b$.

All processes in $\mathscr{P}$ trivially type after the transition.

By hypothesis:

$$\frac{\begin{array}{l} \forall S' \text{ s.t. } S \preceq_L S',\ (S' \models b \Rightarrow \rho, V, L, S' \Vdash P_1) \wedge \\ (S' \models \neg b \Rightarrow \rho, V, L, S' \Vdash P_2) \end{array}}{\rho, V, L, S \Vdash \mathbf{if}\ b\ \mathbf{then} P_1 \mathbf{else}\ P_2}$$

and since $S \preceq_L S$ and $S \models B$ we obtain that $\rho, V, L, S \Vdash P_1$.

Case IF-2.
$\rho, S, \mathscr{P} \uplus \{(\textbf{if } b \textbf{ then } P_1 \textbf{ else } P_2, L, V)\} \rightarrow$
$\rho, S, \mathscr{P} \uplus \{(P_2, L, V)\}$ and $S \not\models b$.

Similarly to the case IF-1, all processes in $\mathscr{P}$ type after the transition, and $\rho, V, L, S \Vdash P_2$.

Case LOCK.
$\rho, S, \mathscr{P} \uplus \{(\textbf{lock}(s); P, L, V)\} \rightarrow \rho, S, \mathscr{P} \uplus \{(P, L \cup \{s\}, V)\}$ and $\forall\, (P', L', V') \in \mathscr{P}\ .\ s \notin L'$

All processes in $\mathscr{P}$ trivially type after the transition.

By hypothesis:

$$\frac{\forall S' \text{ s.t. } S \preceq_L S'\ \rho, V, (L \cup \{s\}), S' \Vdash P}{\rho, V, L, S \Vdash \textbf{lock}(s); P}$$

and in particular since $S \preceq_L S$ we obtain that $\rho, V, L \cup \{s\}, S \Vdash P$.

Case UNLOCK.
$\rho, S, \mathscr{P} \uplus \{(\textbf{unlock}(s); P, L, V)\} \rightarrow \rho, S, \mathscr{P} \uplus \{(P, L \setminus \{s\}, V)\}$ and $s \in L$

All processes in $\mathscr{P}$ trivially type after the transition.

By hypothesis:

$$\frac{\forall S' \text{ s.t. } S \preceq_L S'\ \rho, V, (L \setminus \{s\}), S' \Vdash P}{\rho, V, L, S \Vdash \textbf{unlock}(s); P}$$

and in particular since $S \preceq_L S$ we obtain that $\rho, V, (L \setminus \{s\}), S \Vdash P$.

Case SET.
Let $\rho, S, \mathscr{P} \uplus \{(\textbf{set}(b^+); P_1, L_1, V_1)\} \rightarrow$
$\rho, S', \mathscr{P} \uplus \{(P_1, L_1, V_1)\}$ where $S' = \mathit{update}(S, \rho(b^+))$.

Because by hypothesis:

$$\frac{\begin{array}{c}\forall S_1' \text{ s.t. } S \preceq_{L_1} S_1' (\forall M \in \mathit{fv}(b^+) \cup \mathit{fn}(b^+) \\ \quad \mathsf{implies}(\langle\!|\rho(M)|\!\rangle_{S_1'}, \langle\!|\rho(M)|\!\rangle_{S_1''}) \in \mathscr{F}_{P_0}) \wedge \\ \quad (\forall S'' \text{ s.t. } S_1'' \preceq_{L_1} S_1''',\ \rho, V_1, L_1, S_1''' \Vdash P_1)\end{array}}{\rho, V_1, L_1, S \Vdash \textbf{set}(b^+); P_1}$$

where $S_1'' = \mathit{update}(S_1', \rho(b^+))$, and because $S \preceq_{L_1} S$ and $S' \preceq_{L_1} S'$ then $\rho, V_1, L_1, S' \Vdash P_1$.

Now let (P_2, V_2, L_2) be a tuple in $\mathscr{P}$. We should prove that the typing $\rho, V_2, L_2, S' \Vdash P_2$ holds assuming $\rho, V_2, L_2, S \Vdash P_2$. By Lemma 9 we know that $S \preceq_{L_2} S'$, since $L_1 \cap L_2 = \emptyset$ and for all $M \in fv(b^+) \cup fn(b^+)$ holds $\mathsf{implies}(\{\!|\rho(M)|\!\}_S, \{\!|\rho(M)|\!\}_{S'}) \in \mathscr{F}_{P_0}$. The proof is carried as a case-by-case analysis of the process P_2. The case T-NIL trivially holds after the transition.

For the cases T-PAR, T-REPL, T-IN, T-OUT, T-NEW, T-LET, T-IF, T-LOCK, T-UNLOCK, T-SET the following consideration holds: because the condition of the typing rule $\rho, V_2, L_2, S \Vdash P_2$ is quantified over all states S'' such that $S \preceq_{L_2} S''$, then we have that the states that satisfy $S' \preceq_{L_2} S''$ also satisfy $S \preceq_{L_2} S''$, because $\preceq_{L_2}$ is an order relation and $S \preceq_{L_2} S'$, hence $\rho, V_2, L_2, S' \Vdash P_2$.

APPENDIX B

Carried out examples in Set-Pi

B.1 Key Registration

Here we present a key-registration protocol where an honest principal A registers its current pair of asymmetric keys (pk_A, sk_A) to the server S. An initial pair of keys is distributed securely to A and S, where A knows both public and secret keys while S only knows the public key.

Later in the protocol, before the current key expires, A registers a new key to the server by sending the following message:

$$A \rightarrow S : aenc(sk, \langle new, a, pk' \rangle)$$

which encodes the new public key pk' with the old secret key sk. S will be able to decrypt A's message with the old public key pk, move pk from the set of valid keys to the database of revoked keys and send back an acknowledgment to A.

$$S \rightarrow A : aenc(pk', confirm)$$

In turn A will be able to decrypt this message with sk' and remove the old sk from its key-ring.

$$
\begin{aligned}
A \triangleq\ & \mathbf{in}(kdb_a, sk_a : SKey);\\
& \mathbf{if}\ sk_a \in ring_a\ \mathbf{then}\\
& \mathbf{new}\ s'_a : Seed;\\
& \mathbf{set}(sk(s'_a) \in ring_a);\\
& \mathbf{out}(ch, aenc(sk_a, \langle new\_key, a, pk(s'_a)\rangle));\\
& \mathbf{out}(kdb_a, sk(s'_a));\\
& \mathbf{in}(ch, x_c : aenc(PKey, x_t));\\
& \mathbf{let}\ x_r = adec(sk(s'_a), x_c)\ \mathbf{in}\\
& \mathbf{if}\ x_r = confirm\ \mathbf{then}\\
& \mathbf{set}(\neg sk_a \in ring_a);\\
& \mathbf{out}(ch, sk_a); 0
\end{aligned}
$$

$$
\begin{aligned}
S \triangleq\ & \mathbf{in}(ch, x_s : aenc(SKey, \langle x_t, x_{t'}, PKey\rangle));\\
& \mathbf{in}(kdb_s, pk_a : PKey);\\
& \mathbf{let}\ (= new, = a, pk'_a) = adec(pk_a, xs)\ \mathbf{in}\\
& \mathbf{if}\ pk_a \in valid_a \wedge pk'_a \notin valid_a \wedge pk'_a \notin revoked_a\ \mathbf{then}\\
& \mathbf{set}(pk_a \in revoked_a; \neg pk_a \in valid_a; pk'_a \in valid_a);\\
& \mathbf{out}(ch, aenc(pk'_a, confirm));\\
& \mathbf{out}(kdb_s, pk'_a); 0
\end{aligned}
$$

$$
\begin{aligned}
Sys \triangleq\ & \mathbf{reduc}\ \forall x : Seed, m : t\ .\\
& \qquad adec(pk(x), aenc(sk(x), m)) \rightarrow m;\\
& \mathbf{reduc}\ \forall x : Seed\ .\ keypair(sk(x), pk(x)) \rightarrow true;\\
& \mathbf{new}\ ring_a : \mathbf{set}\ sk(Seed);\\
& \mathbf{new}\ valid_a : \mathbf{set}\ pk(Seed);\\
& \mathbf{new}\ revoked_a : \mathbf{set}\ pk(Seed);\\
& \mathbf{new}\ kdb_a : SKey; \mathbf{new}\ kdb_s : SKey; \mathbf{new}\ s_a : Seed;\\
& \mathbf{set}(sk(s_a) \in ring_a; pk(s_a) \in valid_a);\\
& \mathbf{out}(kdb_a, sk(s_a));\\
& \mathbf{out}(kdb_s, pk(s_a));\\
& (!^{\{ring_a\}} A \mid !^{\{valid_a, revoked_a\}} S)
\end{aligned}
$$

Once a new key is established and the client receives confirmation from the server that the secret key *sk* has been revoked, *sk* can be revealed to the attacker. An attacker

succeeds in breaking the protocol when she discovers a secret key that is still registered to the server.

B.2 Yubikey

Yubikey is a small USB token used to authenticate to supported online services. It works by maintaining a pair of a secret identity (shared with the server) and a public identity (shared publicly), and by sending to the Yubikey server its own public identity, together with the one time password encrypted with the current value of a counter using a shared key k.

Here we model a simplified version of the Yubikey protocol, where we are interested in the injective agreement between the client Yubikey (YK) and the server (Srv). The process BP represents the process activated by pressing they Yubikey button, which authenticates the user to the server. We define a public channel ch, and a private channel ch_server that is only used to securely exchange the secret identity and shared key to the server.

The Yubikey process YK creates a new fresh key k, its own public and secret identities (x_{pid} and x_{sid}), stores them securely to the server, then reveals its public identity and starts the BP process.

The button press (BP) process initiates the authentication procedure, increasing the counter (this is encoded in our calculus by the creation of a fresh value), producing the nonces x_{nonce} and x_{tpr}, and sending the encrypted message. An event yk_press is inserted to denote that the button has been pressed.

The server on the other end receives the login request from the Yubikey, retrieves its secret identity and key k from its own channel, pattern matching on the Yubikey's public identity to find the right tuple, decrypts the message with the retrieved key k, and finally if the counter has not been used, it issues a yk_login event to conclude the protocol.

$$
\begin{aligned}
Sys \triangleq\ & \textbf{new}\ yk\_press : \textbf{event}(cnt);\\
& \textbf{new}\ yk\_login : \textbf{event}(cnt);\\
& \textbf{new}\ used : \textbf{set}\ cnt;\\
& \textbf{reduc}\ \forall x : t, k : key\ .\ sdec(k, senc(k, x)) \rightarrow x;\\
& \textbf{new}\ ch : channel;\\
& \textbf{new}\ ch\_server : channel;
\end{aligned}
$$

$$(!\,YK \mid !\,Srv)$$

$$\begin{aligned}
BP \triangleq\ & \textbf{new}\ x_c : cnt; \\
& \textbf{new}\ x_{nonce} : nonce; \\
& \textbf{new}\ x_{tpr} : nonce; \\
& \textbf{event}\ yk_press(x_c); \\
& \textbf{out}(ch, \langle x_{pid}, x_{nonce}, senc(k, \langle x_{sid}, x_c, x_{tpr}\rangle)\rangle); 0
\end{aligned}$$

$$\begin{aligned}
YK \triangleq\ & \textbf{new}\ k : key; \\
& \textbf{new}\ x_{pid} : pid; \\
& \textbf{new}\ x_{sid} : sid; \\
& \textbf{out}(ch_server, \langle x_{pid}, x_{sid}, k\rangle); \\
& \textbf{out}(ch, x_{pid}); \\
& !\,BP
\end{aligned}$$

$$\begin{aligned}
Srv \triangleq\ & \textbf{in}(ch, \langle x_{pid}, x_{nonce}, x_{enc}\rangle : \\
& \qquad \langle pid, nonce, senc(key, \langle sid, cnt, nonce\rangle)\rangle); \\
& \textbf{in}(ch_server, \langle = x_{pid}, x_{sid}, x_k\rangle : \langle pid, sid, key\rangle); \\
& \textbf{let}\ \langle = x_{sid}, x_{cnt}, x_{tpr}\rangle = sdec(x_k, x_{enc})\ \textbf{in} \\
& \textbf{lock}(used); \\
& \textbf{if}\ x_{cnt} \notin used\ \textbf{then} \\
& \textbf{set}(x_{cnt} \in used); \\
& \textbf{event}\ yk_login(x_{cnt}); \\
& \textbf{unlock}(used); 0
\end{aligned}$$

Here we find an injective agreement between the events *yk_press* and *yk_login*. Although this example shows only one Yubikey and Server pair, it can be extended by including multiple copies of the client and server processes, and copies of the respective sets and events to prove the injective agreement with a finite number of participants.

B.3 Set-Pi Models for the Case Study

```
(* Message Authentication in MaCAN *)

type seed.
type ts = ts(seed).
fun ts(seed).
type msg = msg(seed).
fun msg(seed).

type key.
type channel.

set recent : ts.
set old : ts.
event send(msg).
injevent accept(msg).

fun sign(t,key).
reduc checksign(x, sign(x, y), y) = x.

query x_seed: seed; injagree accept(msg(x_seed)) ⟹ send(msg(x_seed)).

new ch : channel.
new chclock : channel.
new k : key [private].

let A = new s: seed;
        let t: ts = ts(s) in
        let m: msg = msg(s) in
        lock(recent,old);
        set t in recent;
        event send(m);
        out(ch, (t, m, sign((t, m), k)));
        out(chclock, t);
        unlock(recent,old).

let B = in(ch, (xt: ts, xm: msg, xs:sign((ts, msg), key)));
        let xr : msg = checksign((xt, xm), xs, k) in
        lock(recent,old);
        if not xt in old then
            injevent accept(xm);
            unlock(recent,old).
```

```
let Clock = in(chclock, t: ts); set (not t in recent) and (t in old).

process (!A()) | (!B()) | (!{recent,old}Clock())

(* MaCAN TimeServer Fix *)

type seed.
type channel.
type time.

fun pk(seed).
fun sk(seed).

set skewed_time: time.
set checked_time: time.
set old_time: time.
event send(time).
injevent accept(time).

fun aes_enc(t,sk(s)).
reduc aes_dec(aes_enc(x, sk(s)), pk(s)) = x.

query x_t: time; injagree accept(x_t) ⟹ send(x_t).

new ch : channel.

let TS(pk, sk) =
    new t: time;
    event send(t);
    out(ch, (t, aes_enc(t, sk))).

let ECU(pk_ts) =
    lock(skewed_time, checked_time, old_time);
    in(ch, (x_t: time, x_e: aes_enc(time, sk(seed))));
    if (x_t in skewed_time) and (not x_t in old_time) then
        let =x_t: time = aes_dec(x_e, pk_ts) in
        set (not x_t in skewed_time) and (x_t in checked_time);
        injevent accept(x_t);
        unlock(skewed_time, old_time).

let Clock() =
    in(ch,(x_t: time, x_e: aes_enc(time, sk(seed))));
    ((lock(skewed_time, checked_time);
```

```
        if (not x_t in checked_time) then
          set (x_t in skewed_time)) |
      (lock(old_time); set(x_t in old_time))).

process !(new s: seed;
            let pk_ts:pk(seed) = pk(s) in
            let sk_ts:pk(seed) = sk(s) in
            !(TS(pk_ts, sk_ts) | ECU(pk_ts)) | Clock())

(* Key establishment in CANAuth *)

type cnt.

type key.
type channel.

set received : cnt.
event send(cnt).
injevent accept(cnt).

fun hmac(t,t').
type sesskey = hmac(cnt, key).
type keysig = hmac(cnt, sesskey).
reduc checksign(x, hmac(x, y), y) = x.
reduc eq(x, x) = x.

query x_cnt:cnt; injagree accept(x_cnt) ⟹ send(x_cnt).

new ch : channel.
new kp : key [private].

let A = new c: cnt;
        let ks:sesskey = hmac(c, kp) in
        let s1:keysig = hmac(c, ks) in
        event send(c);
        out(ch, c);
        out(ch, s1).

let B = in(ch, x_c: cnt);
        in(ch, x_s1: keysig);
        let ks:sesskey = hmac(x_c, kp) in
        let s2:keysig = hmac(x_c, ks) in
        let =x_s1:keysig = s2 in
        if not x_c in received then
```

```
            set x_c in received;
            injevent accept(x_c).

process (!A()) | (!{received}B())

(* Message authentication in CANAuth *)

type cnt.
fun msg(cnt).
type msg = msg(cnt).
reduc getCnt(msg(x)) = x.

type key.
type channel.

set received : cnt.
event send(msg).
injevent accept(msg).

fun sign(t,key).
reduc checksign(x, sign(x, y), y) = x.

query x_cnt:cnt; injagree accept(msg(x_cnt)) ⟹ send(msg(x_cnt)).
query m:msg; msg(ch, (m, sign(m, k))) where m in accept_twice.

new ch : channel.
new k : key [private].

let A = new c: cnt;
        let m:msg = msg(c) in
        event send(m);
        out(ch, (m, sign(m, k))).

let B = in(ch, (xm: msg, xs:sign(msg, key)));
        let xc : cnt = getCnt(xm) in
        let xr : msg = checksign(xm, xs, k) in
        if not xc in received then
            set xc in received;
            injevent accept(xm).

process (!A()) | (!{received}B())
```

Bibliography

[AAA+12] Alessandro Armando, Wihem Arsac, Tigran Avanesov, Michele Barletta, Alberto Calvi, Alessandro Cappai, Roberto Carbone, Yannick Chevalier, Luca Compagna, Jorge Cuéllar, Gabriel Erzse, Simone Frau, Marius Minea, Sebastian Mödersheim, David von Oheimb, Giancarlo Pellegrino, Serena Elisa Ponta, Marco Rocchetto, Michaël Rusinowitch, Mohammad Torabi Dashti, Mathieu Turuani, and Luca Viganò. The avantssar platform for the automated validation of trust and security of service-oriented architectures. In *Proceedings of TACAS*, pages 267–282, 2012.

[AC08] Alessandro Armando and Luca Compagna. Sat-based model-checking for security protocols analysis. *Int. J. Inf. Sec.*, 7(1):3–32, 2008.

[AF01] Martín Abadi and Cédric Fournet. Mobile values, new names, and secure communication. In *Proceedings of POPL*, pages 104–115, 2001.

[AM14] Omar Almousa and Sebastian Mödersheim. Alice and bob: Reconciling formal models and implementation. 2014.

[AMRR11] Myrto Arapinis, Loretta Ilaria Mancini, Eike Ritter, and Mark Ryan. Formal analysis of UMTS privacy. *CoRR*, abs/1109.2066, 2011.

[ARR11] Myrto Arapinis, Eike Ritter, and Mark D. Ryan. StatVerif: Verification of Stateful Processes. *2011 IEEE 24th Computer Security Foundations*, pages 33–47, 2011.

[BAF05] Bruno Blanchet, Martín Abadi, and Cédric Fournet. Automated verification of selected equivalences for security protocols. In *Logic in Computer Science, 2005. LICS 2005. Proceedings. 20th Annual IEEE Symposium on*, pages 331–340. IEEE, 2005.

[BBD+05] Chiara Bodei, Mikael Buchholtz, Pierpaolo Degano, Flemming Nielson, and Hanne Riis Nielson. Static validation of security protocols. *Journal of Computer Security*, 13(3):347–390, 2005.

[BBF+11] Jesper Bengtson, Karthikeyan Bhargavan, Cédric Fournet, Andrew D Gordon, and Sergio Maffeis. Refinement types for secure implementations. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 33(2):8, 2011.

[BCEM15] Michele Bugliesi, Stefano Calzavara, Fabienne Eigner, and Matteo Maffei. Affine Refinement Types for Secure Distributed Programming. In *TOPLAS*, 2015.

[BGHB11] Gilles Barthe, Benjamin Grégoire, Sylvain Heraud, and Santiago Zanella Béguelin. Computer-aided security proofs for the working cryptographer. In *Advances in Cryptology–CRYPTO 2011*, pages 71–90. Springer, 2011.

[BGZB09] Gilles Barthe, Benjamin Grégoire, and Santiago Zanella Béguelin. Formal certification of code-based cryptographic proofs. *ACM SIGPLAN Notices*, 44(1):90–101, 2009.

[BKA+15] Alessandro Bruni, Markulf Kohlweiss, Myrto Arapinis, Mark D. Ryan, Eike Ritter, Flemming Nielson, and Riis Nielson Hanne. Proving stateful injective agreement with refinement types. In *4th International Cryptoforma Workshop*, 2015.

[Bla01] Bruno Blanchet. An efficient cryptographic protocol verifier based on prolog rules. In *Proceedings of the 14th IEEE workshop on Computer Security Foundations*, pages 82–96. IEEE Computer Society, 2001.

[Bla02] Bruno Blanchet. From secrecy to authenticity in security protocols. In *Proceedings of SAS*, 2002.

[Bla05] Bruno Blanchet. Security protocols: from linear to classical logic by abstract interpretation. *Inf. Process. Lett.*, 95(5):473–479, 2005.

[Bla07] Bruno Blanchet. Cryptoverif: Computationally sound mechanized prover for cryptographic protocols. In *Dagstuhl seminar "Formal Protocol Verification Applied*, page 117, 2007.

[Bla09] Bruno Blanchet. Automatic verification of correspondences for security protocols. *Journal of Computer Security*, 17(4):363–434, 2009.

[BLV07] Reinder J Bril, Johan J Lukkien, and Wim FJ Verhaegh. Worst-case response time analysis of real-time tasks under fixed-priority scheduling with deferred preemption revisited. In *Real-Time Systems, 2007. ECRTS'07. 19th Euromicro Conference on*, pages 269–279. IEEE, 2007.

[BMNN15] Alessandro Bruni, Sebastian Mödersheim, Flemming Nielson, and Hanne Riis Nielson. Set-Pi: Set Membership Pi-calculus. In *CSF*, 2015.

[BMV05] David Basin, Sebastian Mödersheim, and Luca Vigano. Ofmc: A symbolic model checker for security protocols. *International Journal of Information Security*, 4(3):181–208, 2005.

[BR04] M Bellare and P Rogaway. The Game-Playing Technique. *Computer*, pages 1–29, 2004.

[BS13] Bruno Blanchet and Ben Smyth. Proverif 1.88: Automatic cryptographic protocol verifier, user manual and tutorial, 2013.

[BSNN14] Alessandro Bruni, Michal Sojka, F Nielson, and HR Nielson. Formal Security Analysis of the MaCAN Protocol. In *Integrated Formal Methods*, 2014.

[CB12] David Cadé and Bruno Blanchet. From computationally-proved protocol specifications to implementations. In *Availability, Reliability and Security (ARES), 2012 Seventh International Conference on*, pages 65–74. IEEE, 2012.

[CB13] Vincent Cheval and Bruno Blanchet. Proving more observational equivalences with proverif. In *Principles of Security and Trust*, pages 226–246. Springer, 2013.

[CCS10] Juan Chen, Ravi Chugh, and Nikhil Swamy. Type-preserving compilation of end-to-end verification of security enforcement. In *ACM Sigplan Notices*, volume 45, pages 412–423. ACM, 2010.

[CD09] Véronique Cortier and Stéphanie Delaune. A method for proving observational equivalence. In *Computer Security Foundations Symposium, 2009. CSF'09. 22nd IEEE*, pages 266–276. IEEE, 2009.

[CMK+11] Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, Stefan Savage, Karl Koscher, Alexei Czeskis, Franziska Roesner, and Tadayoshi Kohno. Comprehensive experimental analyses of automotive attack surfaces. *Proceedings of the 20th USENIX conference on Security*, 2011.

[Cre08] Cas JF Cremers. The scyther tool: Verification, falsification, and analysis of security protocols. In *Computer Aided Verification*, pages 414–418. Springer, 2008.

[DBBL07] Robert I Davis, Alan Burns, Reinder J Bril, and Johan J Lukkien. Controller area network (CAN) schedulability analysis: Refuted, revisited and revised. *Real-Time Systems*, 35(3):239–272, 2007.

[DKR07] Stéphanie Delaune, Steve Kremer, and Mark Ryan. Symbolic bisimulation for the applied pi calculus. In *FSTTCS 2007: Foundations of Software Technology and Theoretical Computer Science*, pages 133–145. Springer, 2007.

[Dwo05] Morris J Dworkin. *SP 800-38B. Recommendation for Block Cipher Modes of Operation: the CMAC Mode for Authentication*. National Institute of Standards & Technology, 2005.

[DY83] Danny Dolev and Andrew Yao. On the security of public key protocols. *Information Theory, IEEE Transactions on*, 29(2):198–208, 1983.

[Eva11] Dave Evans. The internet of things: How the next evolution of the internet is changing everything. *CISCO white paper*, 1:14, 2011.

[FG14] Marc Fischlin and Felix Günther. Multi-Stage Key Exchange and the Case of Google's QUIC Protocol. In *CCS*, 2014.

[FKS11] Cédric Fournet, Markulf Kohlweiss, and Pierre-Yves Strub. Modular code-based cryptographic verification. In *CCS*, 2011.

[FMD+00] Thomas Führer, Bernd Müller, Werner Dieterle, Florian Hartwich, Robert Hugel, and Michael Walther. Time triggered communication on CAN (Time Triggered CAN-TTCAN). In *7th international CAN Conference*, 2000.

[FS09] Sibylle B. Fröschle and Graham Steel. Analysing pkcs#11 key management apis with unbounded fresh data. In *Proceedings of ARSPA-WITS*, 2009.

[GGH+12] Benjamin Glas, Jorge Guajardo, Hamit Hacioglu, Markus Ihle, Karsten Wehefritz, and Attila Yavuz. Signal-based automotive communication security and its interplay with safety requirements. In *Proceedings of the 10th Escar Conference on Embedded Security in Cars*, 2012.

[GL08] J. Goubault-Larrecq. Towards producing formally checkable security proofs, automatically. In *Proceedings of CSF*, 2008.

[GMHV12] Bogdan Groza, Pal-Stefan Murvay, Anthony Van Herrewege, and Ingrid Verbauwhede. Libra-can: A lightweight broadcast authentication protocol for controller area networks. In *Cryptology and Network Security, 11th International Conference, CANS 2012, Darmstadt, Germany, December 12-14, 2012. Proceedings*, pages 185–200, 2012.

[Hal05] Shai Halevi. A plausible approach to computer-aided cryptographic proofs. *IACR Cryptology ePrint Archive*, 2005:181, 2005.

[Har12] Florian Hartwich. Can with flexible data-rate. In *13th International CAN Conference (iCC2012), Hambach, Germany*, 2012.

[HMF+02] Florian Hartwich, Bernd Müller, Thomas Führer, Robert Hugel, et al. Timing in the TTCAN Network. In *Proceedings of the 8th International CAN Conference (iCC'02)*, 2002.

[HRS12] Oliver Hartkopp, Cornel Reuber, and Roland Schilling. MaCAN - message authenticated CAN. In *Proceedings of the 10th Escar Conference on Embedded Security in Cars*, 2012.

[HSV11] Anthony Van Herrewege, David Singelee, and Ingrid Verbauwhede. CANAuth-A simple, backward compatible broadcast authentication protocol for CAN bus. In *Proceedings of ECRYPT*, 2011.

[KCB97] Hugo Krawczyk, Ran Canetti, and Mihir Bellare. Hmac: Keyed-hashing for message authentication. 1997.

[KCR+10] Karl Koscher, Alexei Czeskis, Franziska Roesner, Shwetak Patel, Tadayoshi Kohno, Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, and Stefan Savage. Experimental Security Analysis of a Modern Automobile. *2010 IEEE Symposium on Security and Privacy*, pages 447–462, 2010.

[KK14] Steve Kremer and Robert Künnemann. Automated analysis of security protocols with global state. In *Proceedings of Security & Privacy*. IEEE Computer Society Press, 2014.

[LJBNR15] Robert Lychev, Samuel Jero, Alexandra Boldyreva, and Cristina Nita-Rotaru. How Secure and Quick is QUIC? Provable Security and Performance Analyses. In *S&P*, 2015.

[Low96] Gavin Lowe. Breaking and fixing the needham-schroeder public-key protocol using fdr. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 147–166. Springer, 1996.

[Low97] Gavin Lowe. A hierarchy of authentication specifications. In *Computer Security Foundations Workshop, 1997. Proceedings., 10th*, pages 31–43. IEEE, 1997.

[Man15] Loretta Ilaria Mancini. *Formal Verification of Privacy in Pervasive Systems*. PhD thesis, University of Birmingham, 2015.

[Möd10] Sebastian Alexander Mödersheim. Abstraction by set-membership: verifying security protocols and web services with databases. In *Proceedings of the 17th ACM conference on Computer and communications security*, pages 351–360. ACM, 2010.

[Mod12] Paolo Modesti. Verified security protocol modeling and implementation with anbx. 2012.

[MPW92] Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, i. *Information and computation*, 100(1):1–40, 1992.

[MU05] Michael Mitzenmacher and Eli Upfal. *Probability and computing: Randomized algorithms and probabilistic analysis*. Cambridge University Press, 2005.

[MV13] Charlie Miller and Chris Valasek. Adventures in automotive networks and control units. In *DEF CON 21 Hacking Conference. Las Vegas, NV: DEF CON*, 2013.

[MV14] Charlie Miller and Chris Valasek. A survey of remote automotive attack surfaces. *Black Hat USA*, 2014.

[MV15] Charlie Miller and Chris Valasek. Remote exploitation of an unaltered passenger vehicle. *Black Hat USA*, 2015.

[Pro13] SESAMO Project. Specification of safety and security analysi s and assessment techniques. Technical report, ARTEMIS, 2013.

[QPN14] Jose Quaresma, Christian W Probst, and Flemming Nielson. The guided system development framework: Modeling and verifying communication systems. In *Leveraging Applications of Formal Methods, Verification and Validation. Specialized Techniques and Applications*, pages 509–523. Springer, 2014.

[RD82] Dorothy Elizabeth Robling Denning. *Cryptography and data security*. Addison-Wesley Longman Publishing Co., Inc., 1982.

[SCF+13] Nikhil Swamy, Juan Chen, Cédric Fournet, Pierre-Yves Strub, Karthikeyan Bhargavan, and Jean Yang. Secure distributed programming with value-dependent types. *Journal of Functional Programming*, 23(04):402–451, 2013.

[SHK+15] Nikhil Swamy, Catalin Hriţcu, Chantal Keller, Aseem Rastogi, Antoine Delignat-Lavaud, Simon Forest, Karthikeyan Bhargavan, Cédric Fournet, Pierre-Yves Strub, Markulf Kohlweiss, and Jean-Karim Zinzindohoue. Dependent types and multi-monadic effects in F*. Draft, July 2015.

[Sho04] Victor Shoup. Sequences of games: a tool for taming complexity in security proofs. *IACR Cryptology ePrint Archive*, 2004:332, 2004.

[SMCB13] B. Schmidt, S. Meier, C. Cremers, and D. Basin. The tamarin prover for the symbolic analysis of security protocols. In *Proceedings of CAV*, 2013.

[Smi14] Craig Smith. *Car Hacker's Manual*. Theia Labs, 2014.

[SRW+11] Hendrik Schweppe, Yves Roudier, Benjamin Weyl, Ludovic Apvrille, and Dirk Scheuermann. Car2x communication: securing the last meter-a cost-effective approach for ensuring trust in car2x applications using in-vehicle symmetric cryptography. In *Vehicular Technology Conference (VTC Fall)*, pages 1–5. IEEE, 2011.

[TH05] Benjamin Tobler and Andrew CM Hutchison. Generating network security protocol implementations from formal specifications. In *Certification and Security in Inter-Organizational E-Service*, pages 33–54. Springer, 2005.

[The03] The AVISPA Project. AVISPA Project Deliverable 2.3: The Intermediate Format, 2003. `http://www.avispa-project.org/publications.html`.

[THW94] KW Tindell, Hans Hansson, and Andy J Wellings. Analysing real-time communications: controller area network (CAN). In *Real-Time Systems Symposium, 1994., Proceedings.*, pages 259–263. IEEE, 1994.

[Tro92] A. S. Troelstra. Tutorial on linear logic. In Krzysztof R. Apt, editor, *Logic Programming, Proceedings of the Joint International Conference and Symposium on Logic Programming, JICSLP 1992, Washington, DC, USA, November 1992*, pages 30–31. MIT Press, 1992.

[VNN15] Roberto Vigo, Hanne Riis Nielson, and Flemming Nielson. *Availability by Design: A Complementary Approach to Denial-of-Service*. PhD thesis, 2015.

[Wei99] Christoph Weidenbach. Towards an automatic analysis of security protocols in first-order logic. In *Proceedings of CADE*, pages 314–328, 1999.

[ZWT09] Tobias Ziermann, Stefan Wildermann, and Jürgen Teich. CAN+: A new backward-compatible Controller Area Network (CAN) protocol with up to 16× higher data rates. In *Design, Automation & Test in Europe Conference & Exhibition, 2009.*, pages 1088–1093. IEEE, 2009.